МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Прикладные системы и фреймворки искусственного интеллекта»

Группа: М8О-403Б-22

Студент: Ибрагимов Р.Р.

Преподаватель:

Москва, 2025

Лабораторная работа №1 (Проведение исследований с алгоритмом KNN)

1. Выбор начальных условий

Выбор набора данных для задачи классификации

Был выбран датасет Sloan Digital Sky Survey DR14.

Автоматическая классификация астрономических объектов (звезды, галактики, квазары) на основе их спектральных характеристик важна для обработки больших объемов данных, генерируемых современными телескопами.

Выбор набора данных для задачи регрессии

Был выбран датасет Abalone Dataset.

Морское ушко — это ценный моллюск, возраст которого напрямую определяет его стоимость и зрелость для вылова. Прямое определение возраста — трудоемкая и разрушающая процедура, требующая разрезания раковины. Предсказание возраста на основе легко измеримых характеристик является неразрушающим и быстрым методом для аквакультуры и экологического мониторинга.

Выбор метрик качества

Для классификации:

- Accuracy (доля правильно предсказанных объектов среди всех, полезна для общей оценки модели на сбалансированных данных)
- Precision (показывает, насколько можно доверять положительному прогнозу модели)
- Recall (показывает, какую долю объектов этого класса модель смогла обнаружить)
- F1-Score (позволяет получить агрегированную оценку по всем классам, полезна на несбалансированных данных)
- Матрица ошибок (наглядное представление того, какие классы путает модель)

Для регрессии:

- MAE (средняя величина абсолютных ошибок в единицах целевой переменной)
- MSE (среднее значение квадратов ошибок, сильнее штрафует за большие ошибки)
- RMSE (как и MSE, более чувствительна к ошибкам, но измеряется в единицах целевой переменной)
- R^2 (оценивает, насколько модель лучше тривиального предсказания среднего возраста)

2. Создание бейзлайна и оценка качества

Классификация

Загрузим датасет и посмотрим на данные

In [1]:
```
import pandas as pd

df_class = pd.read_csv('data/Skyserver_SQL2_27_2018 6_51_39 PM.csv')

print(f"Размерность данных")
print(df_class.shape)
print(f"\nИнформация о данных")
print(df_class.info())
print(f"\nПервые 5 строк")
print(df_class.head())
print(f"\nРаспределение классов")
print(df_class['class'].value_counts())
```

```
Размерность данных
(10000, 18)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   objid      10000 non-null  float64
 1   ra         10000 non-null  float64
 2   dec        10000 non-null  float64
 3   u          10000 non-null  float64
 4   g          10000 non-null  float64
 5   r          10000 non-null  float64
 6   i          10000 non-null  float64
 7   z          10000 non-null  float64
 8   run        10000 non-null  int64
 9   rerun      10000 non-null  int64
 10  camcol     10000 non-null  int64
 11  field      10000 non-null  int64
 12  specobjid  10000 non-null  float64
 13  class      10000 non-null  object
 14  redshift   10000 non-null  float64
 15  plate      10000 non-null  int64
 16  mjd        10000 non-null  int64
 17  fiberid    10000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 1.4+ MB
None

Первые 5 строк
          objid          ra       dec         u         g         r         i  \
0  1.237650e+18  183.531326  0.089693  19.47406  17.04240  15.94699  15.50342
1  1.237650e+18  183.598370  0.135285  18.66280  17.21449  16.67637  16.48922
2  1.237650e+18  183.680207  0.126185  19.38298  18.19169  17.47428  17.08732
3  1.237650e+18  183.870529  0.049911  17.76536  16.60272  16.16116  15.98233
4  1.237650e+18  183.883288  0.102557  17.55025  16.26342  16.43869  16.55492

          z  run  rerun  camcol  field     specobjid   class  redshift  plate  \
0  15.22531  752    301       4    267  3.722360e+18    STAR -0.000009   3306
1  16.39150  752    301       4    267  3.638140e+17    STAR -0.000055    323
2  16.80125  752    301       4    268  3.232740e+17  GALAXY  0.123111    287
3  15.90438  752    301       4    269  3.722370e+18    STAR -0.000111   3306
4  16.61326  752    301       4    269  3.722370e+18    STAR  0.000590   3306

     mjd  fiberid
0  54922      491
1  51615      541
2  52023      513
3  54922      510
4  54922      512

Распределение классов
class
GALAXY    4998
STAR      4152
QSO        850
Name: count, dtype: int64
```

Выделим исходные признаки, которые непосредственно описывают физические свойства объектов и целевую переменную. А также закодируем таргет, так как это категориальный признак.

In [2]:
```python
from sklearn.preprocessing import LabelEncoder

X_class = df_class[['u', 'g', 'r', 'i', 'z', 'redshift']]
y_class = df_class['class']

le = LabelEncoder()
y_class_encoded = le.fit_transform(y_class)
class_names = le.classes_
```

Разделим данные на выборку для обучения и тестовую выборку.

In [3]:
```python
from sklearn.model_selection import train_test_split

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class_encoded, test_size=0.2, random_state=42
)
```

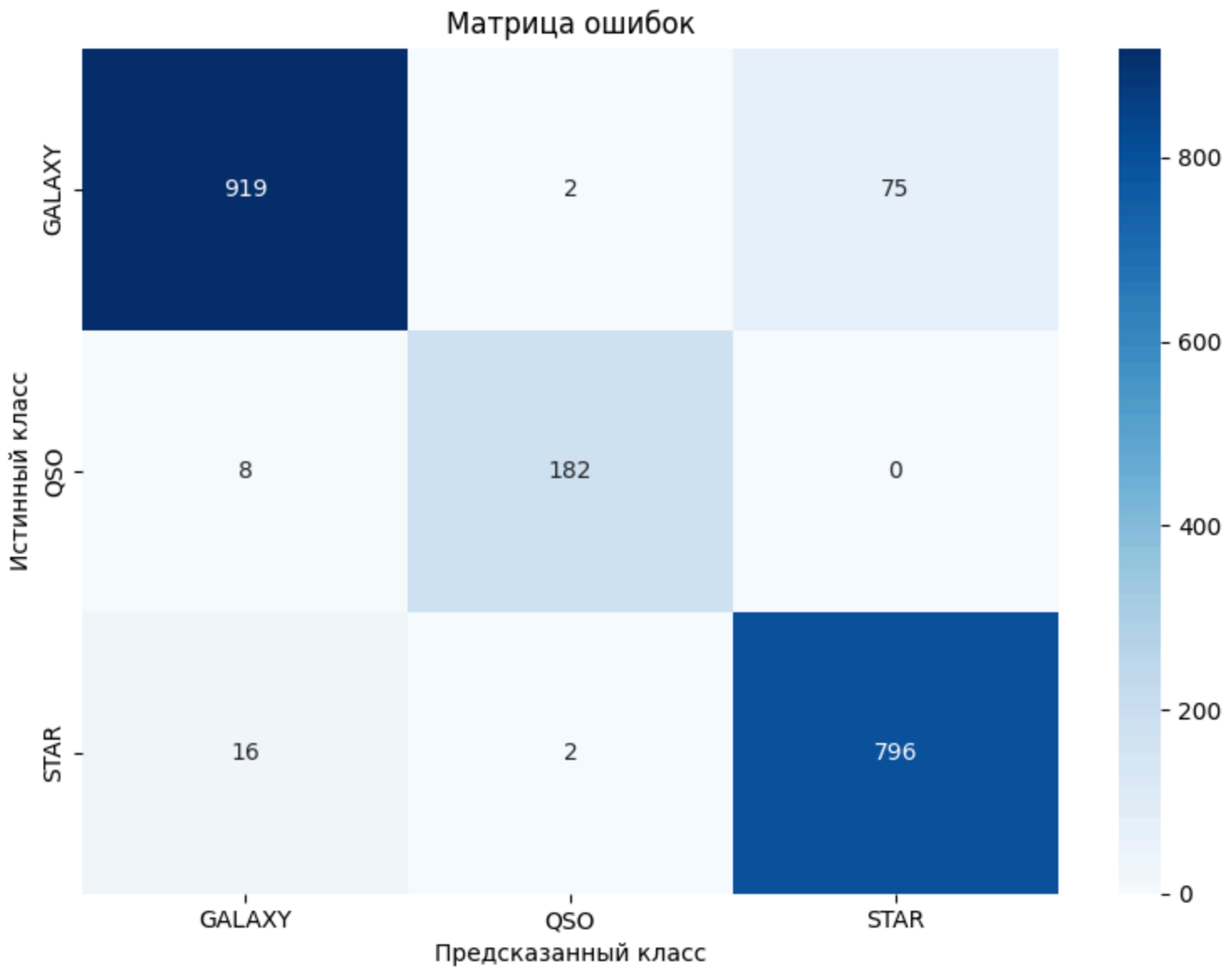
Обучим базовую модель KNN и выполним предсказания на тестовой выборке.

In [4]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_class, y_train_class)

y_pred_class = knn_classifier.predict(X_test_class)
```

Опишем функцию, которая будет использоваться для оценки обученной модели классификации.

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix
)

def evaluate_classification_model(y_true, y_pred, class_names):
    accuracy = accuracy_score(y_true, y_pred)
    print(f"1. Accuracy: {accuracy:.4f}")

    print(f"\n2. Метрики по классам:")
    precision = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)

    metrics_df = pd.DataFrame({
        'Класс': class_names,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
    print(metrics_df.to_string(index=False))

    macro_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"\n3. Macro F1: {macro_f1:.4f}")

    print(f"\n4. Матрица ошибок:")
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
    print(cm_df)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Матрица ошибок')
    plt.ylabel('Истинный класс')
    plt.xlabel('Предсказанный класс')
    plt.tight_layout()
    plt.show()

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'macro_f1': macro_f1,
        'confusion_matrix': cm
    }

class_metrics = evaluate_classification_model(y_test_class, y_pred_class, class_names)
```

```
1. Accuracy: 0.9485

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.974549 0.922691  0.947911
   QSO   0.978495 0.957895  0.968085
  STAR   0.913892 0.977887  0.944807

3. Macro F1: 0.9536

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     919    2    75
QSO          8  182     0
STAR        16    2   796
```



Регрессия

Загрузим датасет и посмотрим на данные

In [6]:
```
df_reg = pd.read_csv('data/abalone.csv')

print(f"Размерность данных")
print(df_reg.shape)
print(f"\nИнформация о данных")
print(df_reg.info())
print(f"\nПервые 5 строк")
print(df_reg.head())
print(f"\nСтатистика по числовым признакам")
print(df_reg.describe())
```

```
Размерность данных
(4177, 9)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None

Первые 5 строк
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7

Статистика по числовым признакам
            Length     Diameter       Height  Whole weight  Shucked weight  \
count  4177.000000  4177.000000  4177.000000   4177.000000     4177.000000
mean      0.523992     0.407881     0.139516      0.828742        0.359367
std       0.120093     0.099240     0.041827      0.490389        0.221963
min       0.075000     0.055000     0.000000      0.002000        0.001000
25%       0.450000     0.350000     0.115000      0.441500        0.186000
50%       0.545000     0.425000     0.140000      0.799500        0.336000
75%       0.615000     0.480000     0.165000      1.153000        0.502000
max       0.815000     0.650000     1.130000      2.825500        1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Выделим признаки и целевую переменную. Закодируем категориальный признак Sex.

In [7]:
```
X_reg = df_reg.drop('Rings', axis=1)
y_reg = df_reg['Rings']

le_sex = LabelEncoder()
X_reg_encoded = X_reg.copy()
X_reg_encoded['Sex'] = le_sex.fit_transform(X_reg['Sex'])
```

Разделим данные на выборку для обучения и тестовую выборку.

In [8]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg_encoded, y_reg, test_size=0.2, random_state=42
)
```

Обучим базовую модель KNN и выполним предсказания на тестовой выборке.

In [9]:
```
from sklearn.neighbors import KNeighborsRegressor

knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = knn_regressor.predict(X_test_reg)
```

Опишем функцию, которая будет использоваться для оценки обученной модели регрессии.

In [10]:
```
import numpy as np
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
```

```
    r2_score,
)

def evaluate_regression_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    print(f"1. MAE: {mae:.4f}")

    mse = mean_squared_error(y_true, y_pred)
    print(f"\n2. MSE: {mse:.4f}")

    rmse = np.sqrt(mse)
    print(f"\n3. RMSE: {rmse:.4f}")

    r2 = r2_score(y_true, y_pred)
    print(f"\n4. R²: {r2:.4f}")

    return {
        'mae': mae,
        'mse': mse,
        'rmse': rmse,
        'r2': r2
    }

reg_metrics = evaluate_regression_model(y_test_reg, y_pred_reg)
```

```
1. MAE: 1.5560

2. MSE: 5.1315

3. RMSE: 2.2653

4. R²: 0.5260
```

3. Улучшение бейзлайна

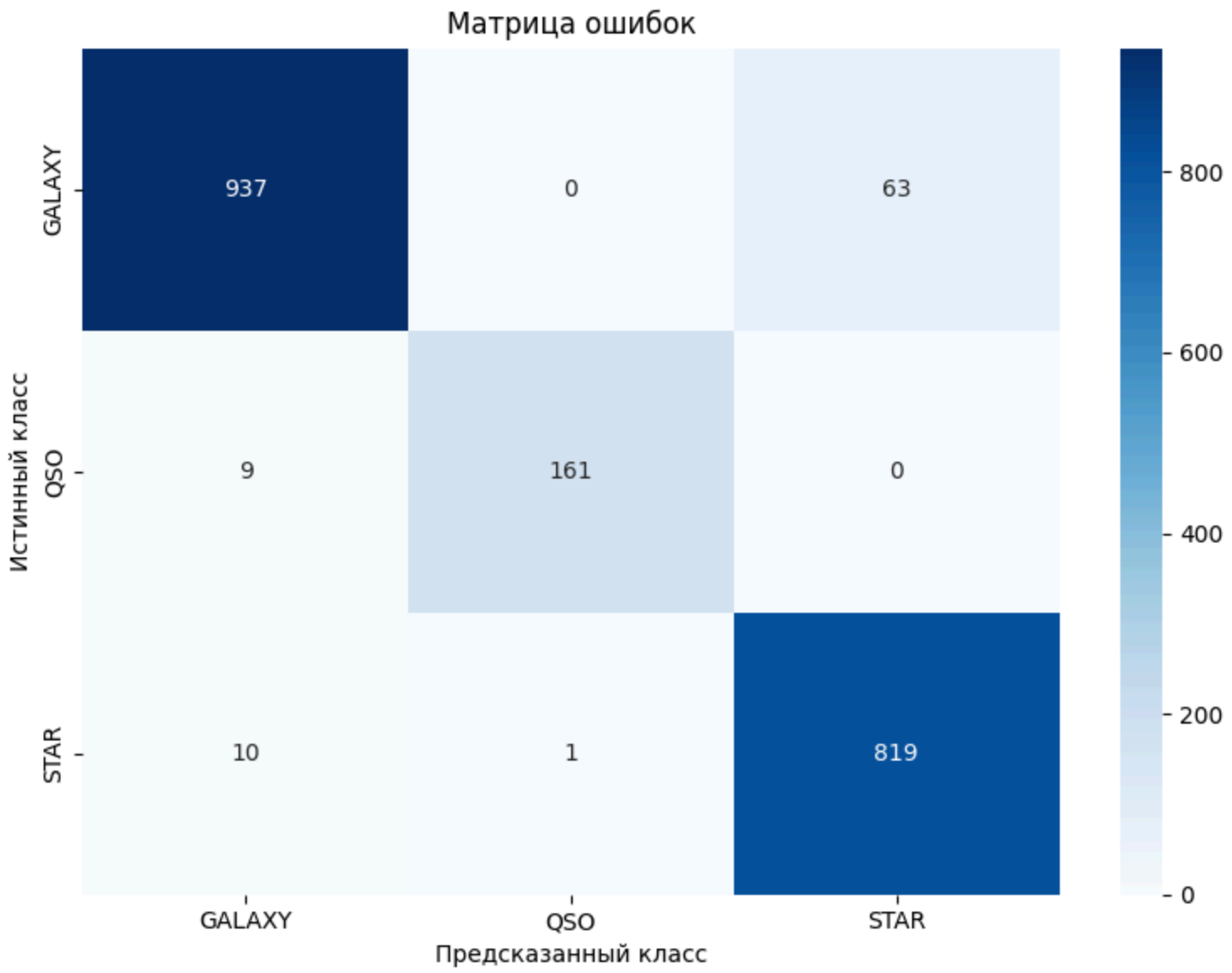
Перейдем к формулированию и проверкам гипотез

Классификация

Гипотеза 1: Добавление стандартизации признаков и стратификации при разбиении на выборки улучшит качество модели.

In [11]:
```
from sklearn.preprocessing import StandardScaler

X_train_class_h1, X_test_class_h1, y_train_class_h1, y_test_class_h1 = train_test_split(
    X_class, y_class_encoded, test_size=0.2, stratify=y_class_encoded, random_state=42
)

scaler = StandardScaler()
X_train_class_h1_scaled = scaler.fit_transform(X_train_class_h1)
X_test_class_h1_scaled = scaler.transform(X_test_class_h1)

knn_classifier_h1 = KNeighborsClassifier(n_neighbors=5)
knn_classifier_h1.fit(X_train_class_h1_scaled, y_train_class_h1)

y_pred_class_h1 = knn_classifier_h1.predict(X_test_class_h1_scaled)

print("Результаты гипотезы 1:")
class_metrics_h1 = evaluate_classification_model(y_test_class_h1, y_pred_class_h1, class_names)
```

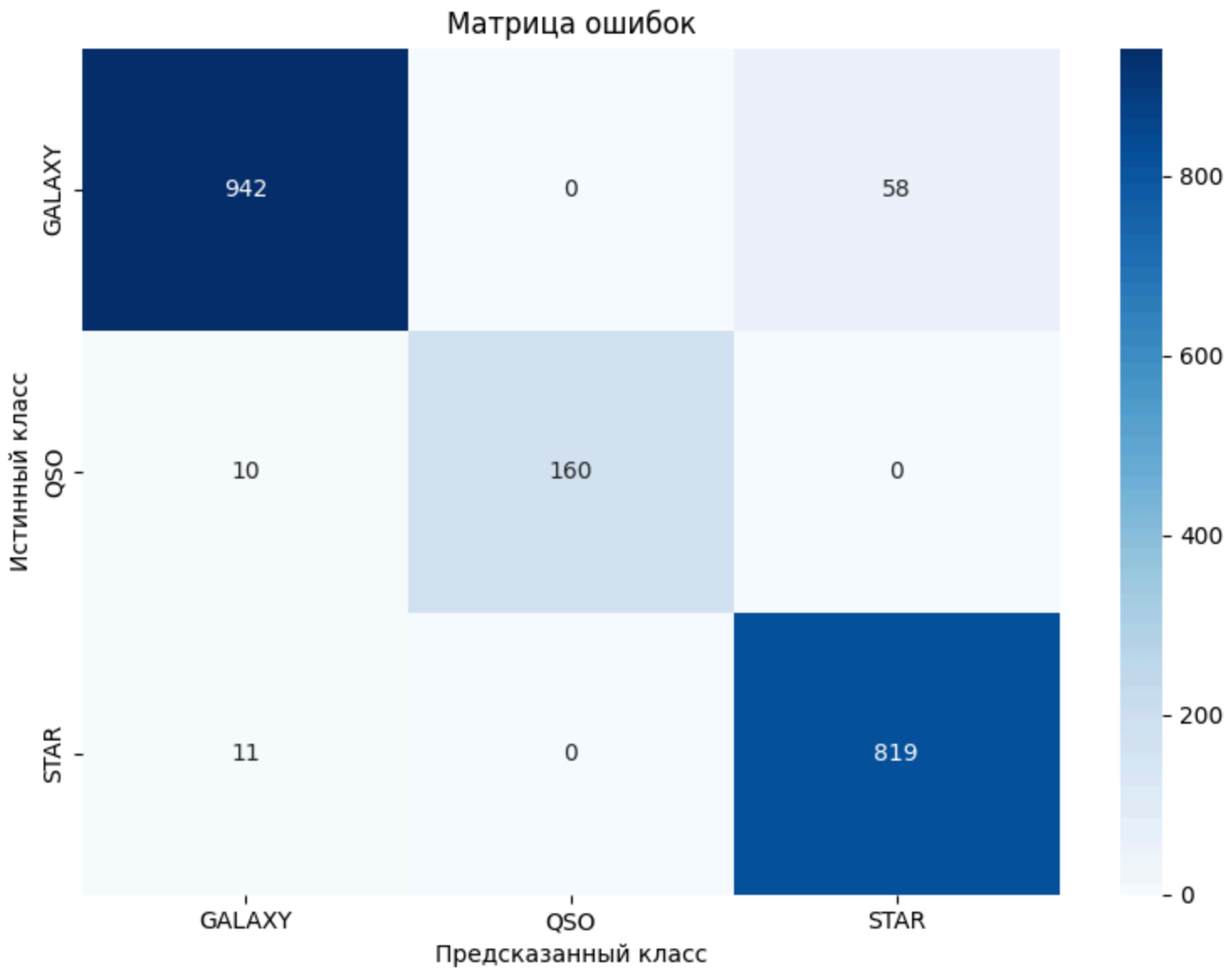
```
Результаты гипотезы 1:
1. Accuracy: 0.9585

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.980126 0.937000  0.958078
   QSO   0.993827 0.947059  0.969880
  STAR   0.928571 0.986747  0.956776

3. Macro F1: 0.9616

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     937    0    63
QSO          9  161     0
STAR        10    1   819
```



Гипотеза 2: Подбор оптимального числа соседей на кросс-валидации улучшит качество модели.

In [12]:
```python
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(3, 21, 2)}
grid_search_h2 = GridSearchCV(
    KNeighborsClassifier(),
    param_grid,
    cv=5,
    scoring='f1_macro',
    n_jobs=-1
)
grid_search_h2.fit(X_train_class_h1_scaled, y_train_class_h1)

print(f"Лучший параметр k: {grid_search_h2.best_params_['n_neighbors']}")
print(f"Лучший F1-score на кросс-валидации: {grid_search_h2.best_score_:.4f}")

knn_classifier_h2 = grid_search_h2.best_estimator_
y_pred_class_h2 = knn_classifier_h2.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 2:")
class_metrics_h2 = evaluate_classification_model(y_test_class_h1, y_pred_class_h2, class_names)
```

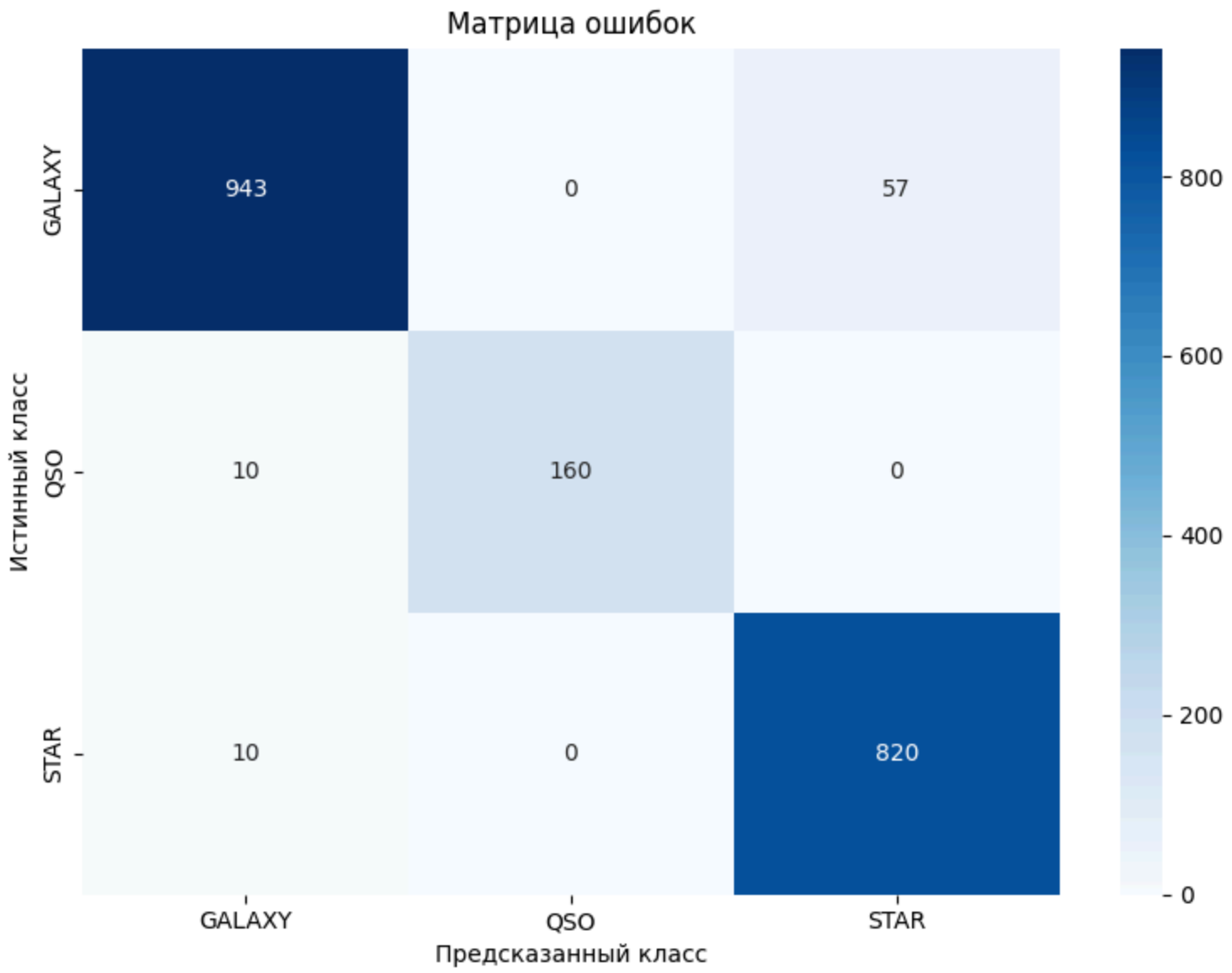
```
Лучший параметр k: 3
Лучший F1-score на кросс-валидации: 0.9585

Результаты гипотезы 2:
1. Accuracy: 0.9605

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.978193 0.942000  0.959755
   QSO   1.000000 0.941176  0.969697
  STAR   0.933865 0.986747  0.959578

3. Macro F1: 0.9630

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     942    0    58
QSO         10  160     0
STAR        11    0   819
```



Гипотеза 3: Использование взвешенных расстояний улучшит качество модели.

In [13]:
```python
param_grid_h3 = {
    'n_neighbors': [grid_search_h2.best_params_['n_neighbors']],
    'weights': ['uniform', 'distance']
}
grid_search_h3 = GridSearchCV(
    KNeighborsClassifier(),
    param_grid_h3,
    cv=5,
    scoring='f1_macro',
    n_jobs=-1
)
grid_search_h3.fit(X_train_class_h1_scaled, y_train_class_h1)

print(f"Лучшие параметры: {grid_search_h3.best_params_}")
print(f"Лучший F1-score на кросс-валидации: {grid_search_h3.best_score_:.4f}")

knn_classifier_h3 = grid_search_h3.best_estimator_
y_pred_class_h3 = knn_classifier_h3.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 3:")
class_metrics_h3 = evaluate_classification_model(y_test_class_h1, y_pred_class_h3, class_names)
```

```
Лучшие параметры: {'n_neighbors': 3, 'weights': 'distance'}
Лучший F1-score на кросс-валидации: 0.9591

Результаты гипотезы 3:
1. Accuracy: 0.9615

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.979232 0.943000  0.960774
   QSO   1.000000 0.941176  0.969697
  STAR   0.935006 0.987952  0.960750

3. Macro F1: 0.9637

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     943    0    57
QSO         10  160     0
STAR        10    0   820
```



Сравним результаты всех гипотез с базовой моделью

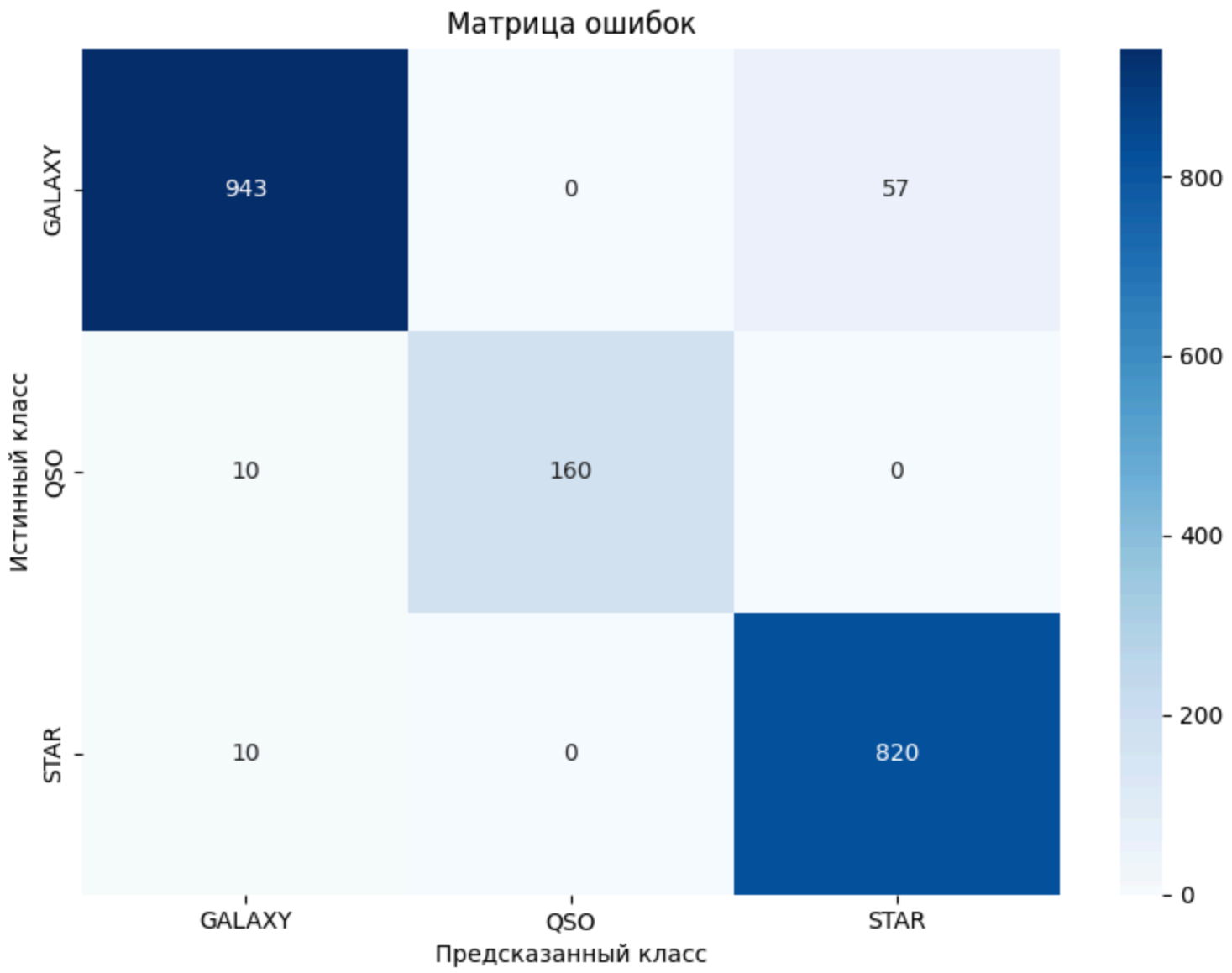
In [14]:
```
comparison_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Гипотеза 1 (стандартизация)',
               'Гипотеза 2 (подбор k)', 'Гипотеза 3 (взвешенные расстояния)'],
    'Accuracy': [class_metrics['accuracy'], class_metrics_h1['accuracy'],
                 class_metrics_h2['accuracy'], class_metrics_h3['accuracy']],
    'Macro F1': [class_metrics['macro_f1'], class_metrics_h1['macro_f1'],
                 class_metrics_h2['macro_f1'], class_metrics_h3['macro_f1']]
})

print("Сравнение моделей классификации:")
print(comparison_class.to_string(index=False))
```

```
Сравнение моделей классификации:
                          Модель  Accuracy  Macro F1
                Базовый бейзлайн    0.9485  0.953601
      Гипотеза 1 (стандартизация)    0.9585  0.961578
            Гипотеза 2 (подбор k)    0.9605  0.963010
Гипотеза 3 (взвешенные расстояния)    0.9615  0.963740
```

Обучим финальную улучшенную модель

In [15]:
```
knn_classifier_improved = KNeighborsClassifier(
    n_neighbors=grid_search_h3.best_params_['n_neighbors'],
    weights=grid_search_h3.best_params_['weights']
)
knn_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_improved = knn_classifier_improved.predict(X_test_class_h1_scaled)

print("Результаты улучшенного бейзлайна для классификации:")
class_metrics_improved = evaluate_classification_model(y_test_class_h1, y_pred_class_improved, class_names
```

```
Результаты улучшенного бейзлайна для классификации:
1. Accuracy: 0.9615

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.979232 0.943000  0.960774
   QSO   1.000000 0.941176  0.969697
  STAR   0.935006 0.987952  0.960750

3. Macro F1: 0.9637

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     943    0    57
QSO         10  160     0
STAR        10    0   820
```



Регрессия

Гипотеза 1: Добавление стандартизации признаков улучшит качество модели.

In [16]:
```
scaler_reg = StandardScaler()
X_train_reg_h1_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_h1_scaled = scaler_reg.transform(X_test_reg)

knn_regressor_h1 = KNeighborsRegressor(n_neighbors=5)
knn_regressor_h1.fit(X_train_reg_h1_scaled, y_train_reg)

y_pred_reg_h1 = knn_regressor_h1.predict(X_test_reg_h1_scaled)

print("Результаты гипотезы 1:")
reg_metrics_h1 = evaluate_regression_model(y_test_reg, y_pred_reg_h1)
```

```
Результаты гипотезы 1:
1. MAE: 1.6084

2. MSE: 5.2379

3. RMSE: 2.2887

4. R²: 0.5161
```

Гипотеза 2: Подбор оптимального числа соседей на кросс-валидации улучшит качество модели.

In [17]:
```
param_grid_reg = {'n_neighbors': range(3, 21, 2)}
grid_search_reg_h2 = GridSearchCV(
    KNeighborsRegressor(),
    param_grid_reg,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search_reg_h2.fit(X_train_reg_h1_scaled, y_train_reg)

print(f"Лучший параметр k: {grid_search_reg_h2.best_params_['n_neighbors']}")
print(f"Лучший MSE на кросс-валидации: {-grid_search_reg_h2.best_score_:.4f}")

knn_regressor_h2 = grid_search_reg_h2.best_estimator_
y_pred_reg_h2 = knn_regressor_h2.predict(X_test_reg_h1_scaled)

print("\nРезультаты гипотезы 2:")
reg_metrics_h2 = evaluate_regression_model(y_test_reg, y_pred_reg_h2)
```

```
Лучший параметр k: 11
Лучший MSE на кросс-валидации: 4.9642

Результаты гипотезы 2:
1. MAE: 1.5731

2. MSE: 5.1318

3. RMSE: 2.2654

4. R²: 0.5259
```

Гипотеза 3: Использование взвешенных расстояний улучшит качество модели.

In [18]:
```
param_grid_reg_h3 = {
    'n_neighbors': [grid_search_reg_h2.best_params_['n_neighbors']],
    'weights': ['uniform', 'distance']
}
grid_search_reg_h3 = GridSearchCV(
    KNeighborsRegressor(),
    param_grid_reg_h3,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search_reg_h3.fit(X_train_reg_h1_scaled, y_train_reg)

print(f"Лучшие параметры: {grid_search_reg_h3.best_params_}")
print(f"Лучший MSE на кросс-валидации: {-grid_search_reg_h3.best_score_:.4f}")

knn_regressor_h3 = grid_search_reg_h3.best_estimator_
y_pred_reg_h3 = knn_regressor_h3.predict(X_test_reg_h1_scaled)

print("\nРезультаты гипотезы 3:")
reg_metrics_h3 = evaluate_regression_model(y_test_reg, y_pred_reg_h3)
```

```
Лучшие параметры: {'n_neighbors': 11, 'weights': 'distance'}
Лучший MSE на кросс-валидации: 4.9392

Результаты гипотезы 3:
1. MAE: 1.5752

2. MSE: 5.1252

3. RMSE: 2.2639

4. R²: 0.5265
```

Сравним результаты всех гипотез с базовой моделью

In [19]:
```
comparison_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Гипотеза 1 (стандартизация)',
               'Гипотеза 2 (подбор k)', 'Гипотеза 3 (взвешенные расстояния)'],
    'MAE': [reg_metrics['mae'], reg_metrics_h1['mae'],
            reg_metrics_h2['mae'], reg_metrics_h3['mae']],
    'RMSE': [reg_metrics['rmse'], reg_metrics_h1['rmse'],
             reg_metrics_h2['rmse'], reg_metrics_h3['rmse']],
    'R²': [reg_metrics['r2'], reg_metrics_h1['r2'],
           reg_metrics_h2['r2'], reg_metrics_h3['r2']]
})

print("Сравнение моделей регрессии:")
print(comparison_reg.to_string(index=False))
```

```
Сравнение моделей регрессии:
                           Модель      MAE     RMSE       R²
                Базовый бейзлайн 1.555981 2.265278 0.525969
      Гипотеза 1 (стандартизация) 1.608373 2.288655 0.516135
            Гипотеза 2 (подбор k) 1.573075 2.265356 0.525937
Гипотеза 3 (взвешенные расстояния) 1.575222 2.263895 0.526548
```

Обучим финальную улучшенную модель

In [20]:
```
knn_regressor_improved = KNeighborsRegressor(
    n_neighbors=grid_search_reg_h3.best_params_['n_neighbors'],
    weights=grid_search_reg_h3.best_params_['weights']
)
knn_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_improved = knn_regressor_improved.predict(X_test_reg_h1_scaled)

print("Результаты улучшенного бейзлайна для регрессии:")
reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_reg_improved)
```

```
Результаты улучшенного бейзлайна для регрессии:
1. MAE: 1.5752

2. MSE: 5.1252

3. RMSE: 2.2639

4. R²: 0.5265
```

4. Имплементация алгоритма машинного обучения

Перейдем к имлпементации алгоритмов

Классификация

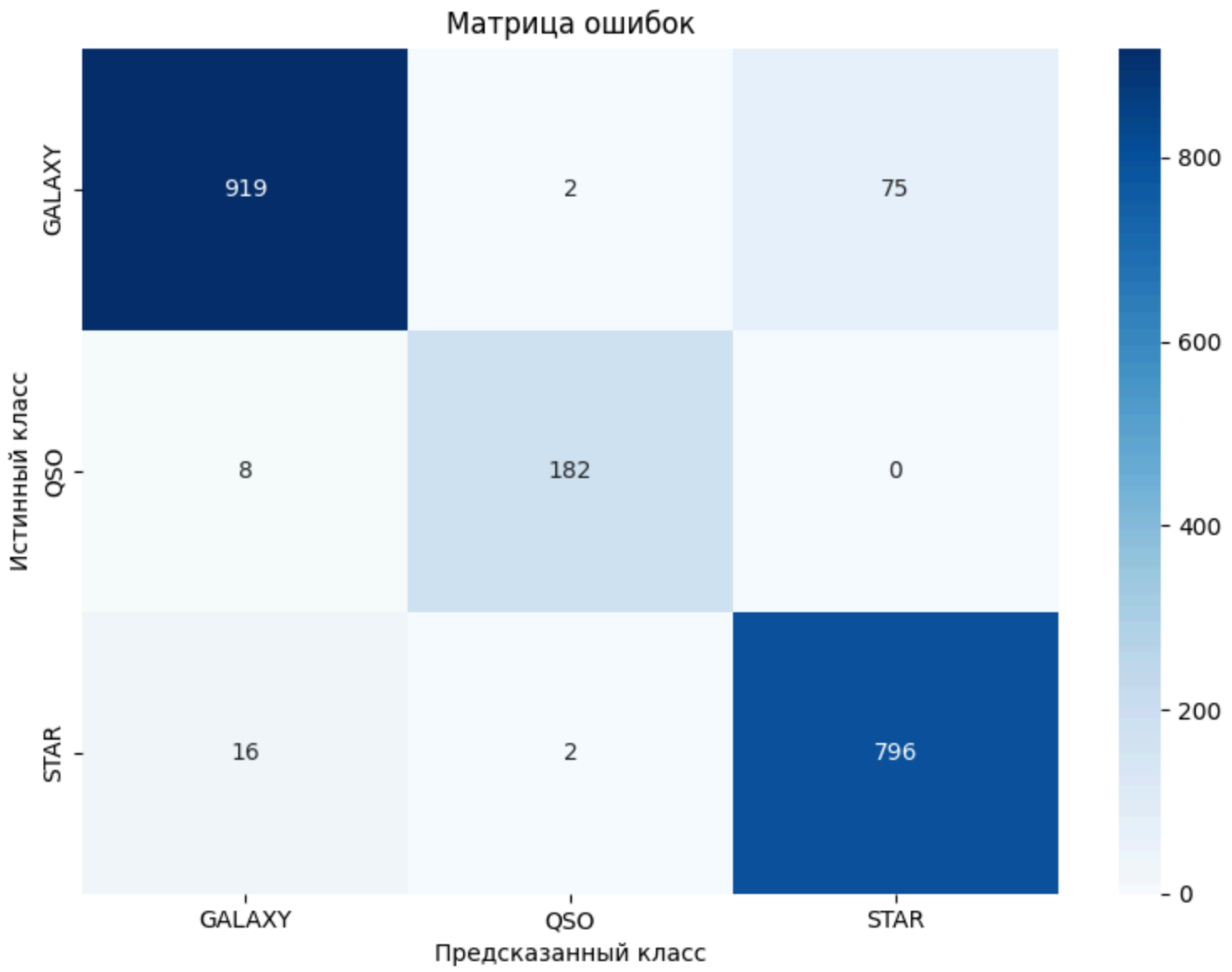
Реализуем алгоритм KNN для классификации

In [21]:
```python
class MyKNNClassifier:
    def __init__(self, n_neighbors=5, weights='uniform'):
        self.n_neighbors = n_neighbors
        self.weights = weights
        self.X_train = None
        self.y_train = None

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        return self

    def _euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))

    def _get_neighbors(self, x):
        distances = []
        for i in range(len(self.X_train)):
            dist = self._euclidean_distance(x, self.X_train[i])
            distances.append((dist, self.y_train[i]))
        distances.sort(key=lambda t: t[0])
        neighbors = distances[:self.n_neighbors]
        return neighbors

    def predict(self, X):
        predictions = []
        X = np.array(X)
        for x in X:
            neighbors = self._get_neighbors(x)
            if self.weights == 'uniform':
                neighbor_labels = [label for _, label in neighbors]
                prediction = max(set(neighbor_labels), key=neighbor_labels.count)
            else:
                neighbor_labels = [label for _, label in neighbors]
                neighbor_distances = [dist for dist, _ in neighbors]
                weights = [1.0 / (d + 1e-10) for d in neighbor_distances]
                label_weights = {}
                for label, weight in zip(neighbor_labels, weights):
                    label_weights[label] = label_weights.get(label, 0) + weight
                prediction = max(label_weights, key=label_weights.get)
            predictions.append(prediction)
        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [22]:
```python
my_knn_classifier = MyKNNClassifier(n_neighbors=5, weights='uniform')
my_knn_classifier.fit(X_train_class.values, y_train_class)
y_pred_my_class = my_knn_classifier.predict(X_test_class.values)

print("Результаты имплементированной модели классификации:")
my_class_metrics = evaluate_classification_model(y_test_class, y_pred_my_class, class_names)
```

```
Результаты имплементированной модели классификации:
1. Accuracy: 0.9485

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.974549 0.922691  0.947911
   QSO   0.978495 0.957895  0.968085
  STAR   0.913892 0.977887  0.944807

3. Macro F1: 0.9536

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     919    2    75
QSO          8  182     0
STAR        16    2   796
```



Сравним результаты имплементированной модели с базовым бейзлайном

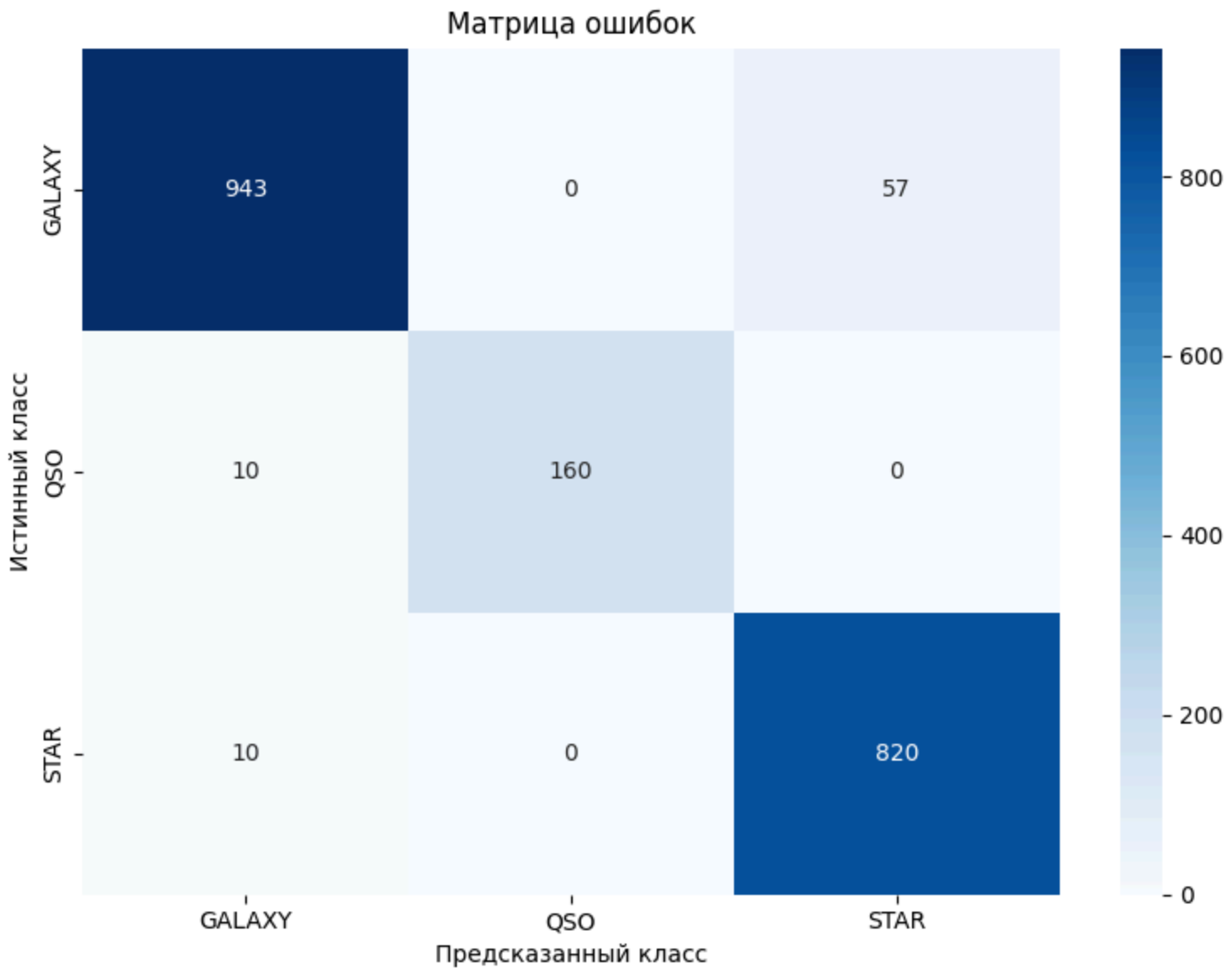
In [23]:
```
comparison_my_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель'],
    'Accuracy': [class_metrics['accuracy'], my_class_metrics['accuracy']],
    'Macro F1': [class_metrics['macro_f1'], my_class_metrics['macro_f1']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_class.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                   Модель  Accuracy  Macro F1
Базовый бейзлайн (sklearn)    0.9485  0.953601
 Имплементированная модель    0.9485  0.953601
```

Теперь применим техники из улучшенного бейзлайна

In [24]:
```
best_k = grid_search_h3.best_params_['n_neighbors']
best_weights = grid_search_h3.best_params_['weights']

my_knn_classifier_improved = MyKNNClassifier(n_neighbors=best_k, weights=best_weights)
my_knn_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_my_class_improved = my_knn_classifier_improved.predict(X_test_class_h1_scaled)

print("Результаты имплементированной модели с улучшениями:")
my_class_metrics_improved = evaluate_classification_model(y_test_class_h1, y_pred_my_class_improved, class_
```

```
Результаты имплементированной модели с улучшениями:
1. Accuracy: 0.9615

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.979232 0.943000  0.960774
   QSO   1.000000 0.941176  0.969697
  STAR   0.935006 0.987952  0.960750

3. Macro F1: 0.9637

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     943    0    57
QSO         10  160     0
STAR        10    0   820
```



Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [25]:
```
comparison_my_class_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'Accuracy': [class_metrics_improved['accuracy'], my_class_metrics_improved['accuracy']],
    'Macro F1': [class_metrics_improved['macro_f1'], my_class_metrics_improved['macro_f1']]
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:")
print(comparison_my_class_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                              Модель  Accuracy  Macro F1
          Улучшенный бейзлайн (sklearn)    0.9615   0.96374
Имплементированная модель с улучшениями    0.9615   0.96374
```

Регрессия

Реализуем алгоритм KNN для регрессии

In [26]:
```
class MyKNNRegressor:
    def __init__(self, n_neighbors=5, weights='uniform'):
        self.n_neighbors = n_neighbors
        self.weights = weights
        self.X_train = None
        self.y_train = None

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        return self

    def _euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))

    def _get_neighbors(self, x):
        distances = []
        for i in range(len(self.X_train)):
            dist = self._euclidean_distance(x, self.X_train[i])
            distances.append((dist, self.y_train[i]))
        distances.sort(key=lambda t: t[0])
        neighbors = distances[:self.n_neighbors]
        return neighbors

    def predict(self, X):
        predictions = []
        X = np.array(X)
        for x in X:
            neighbors = self._get_neighbors(x)
            if self.weights == 'uniform':
                neighbor_values = [value for _, value in neighbors]
                prediction = np.mean(neighbor_values)
            else:
```

```
                neighbor_values = [value for _, value in neighbors]
                neighbor_distances = [dist for dist, _ in neighbors]
                weights = [1.0 / (d + 1e-10) for d in neighbor_distances]
                prediction = np.average(neighbor_values, weights=weights)
            predictions.append(prediction)
        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [27]:
```
my_knn_regressor = MyKNNRegressor(n_neighbors=5, weights='uniform')
my_knn_regressor.fit(X_train_reg.values, y_train_reg.values)
y_pred_my_reg = my_knn_regressor.predict(X_test_reg.values)

print("Результаты имплементированной модели регрессии:")
my_reg_metrics = evaluate_regression_model(y_test_reg, y_pred_my_reg)
```

```
Результаты имплементированной модели регрессии:
1. MAE: 1.5560

2. MSE: 5.1315

3. RMSE: 2.2653

4. R²: 0.5260
```

Сравним результаты имплементированной модели с базовым бейзлайном

In [28]:
```
comparison_my_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель'],
    'MAE': [reg_metrics['mae'], my_reg_metrics['mae']],
    'RMSE': [reg_metrics['rmse'], my_reg_metrics['rmse']],
    'R²': [reg_metrics['r2'], my_reg_metrics['r2']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_reg.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                  Модель      MAE     RMSE       R²
Базовый бейзлайн (sklearn) 1.555981 2.265278 0.525969
 Имплементированная модель 1.555981 2.265278 0.525969
```

Теперь применим техники из улучшенного бейзлайна

In [29]:
```
best_k_reg = grid_search_reg_h3.best_params_['n_neighbors']
best_weights_reg = grid_search_reg_h3.best_params_['weights']

my_knn_regressor_improved = MyKNNRegressor(n_neighbors=best_k_reg, weights=best_weights_reg)
my_knn_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg.values)
y_pred_my_reg_improved = my_knn_regressor_improved.predict(X_test_reg_h1_scaled)

print("Результаты имплементированной модели с улучшениями:")
my_reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_my_reg_improved)
```

```
Результаты имплементированной модели с улучшениями:
1. MAE: 1.5752

2. MSE: 5.1252

3. RMSE: 2.2639

4. R²: 0.5265
```

Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [30]:
```
comparison_my_reg_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'MAE': [reg_metrics_improved['mae'], my_reg_metrics_improved['mae']],
    'RMSE': [reg_metrics_improved['rmse'], my_reg_metrics_improved['rmse']],
    'R²': [reg_metrics_improved['r2'], my_reg_metrics_improved['r2']]
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:")
print(comparison_my_reg_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                                Модель      MAE     RMSE       R²
          Улучшенный бейзлайн (sklearn) 1.575222 2.263895 0.526548
Имплементированная модель с улучшениями 1.575222 2.263895 0.526548
```

Общие выводы по результатам всех моделей

Сравним все 4 модели для классификации и регрессии: базовый бейзлайн из sklearn, имплементированную модель базового бейзлайна, модель с улучшенным бейзлайном из sklearn и имплементированную модель улучшенного бейзлайна.

Классификация

In [31]:
```
final_comparison_class = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'Accuracy': [
        class_metrics['accuracy'],
        my_class_metrics['accuracy'],
        class_metrics_improved['accuracy'],
        my_class_metrics_improved['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        my_class_metrics['macro_f1'],
        class_metrics_improved['macro_f1'],
        my_class_metrics_improved['macro_f1']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ")

print(final_comparison_class.to_string(index=False))
print("\n")

print("Детальное сравнение метрик по классам:")
print("\n1. Базовый бейзлайн (sklearn):")
print(f"   Precision: {class_metrics['precision']}")
print(f"   Recall: {class_metrics['recall']}")
print(f"   F1-score: {class_metrics['f1']}")

print("\n2. Имплементированная модель базового бейзлайна:")
print(f"   Precision: {my_class_metrics['precision']}")
print(f"   Recall: {my_class_metrics['recall']}")
print(f"   F1-score: {my_class_metrics['f1']}")

print("\n3. Улучшенный бейзлайн (sklearn):")
print(f"   Precision: {class_metrics_improved['precision']}")
print(f"   Recall: {class_metrics_improved['recall']}")
print(f"   F1-score: {class_metrics_improved['f1']}")

print("\n4. Имплементированная модель улучшенного бейзлайна:")
print(f"   Precision: {my_class_metrics_improved['precision']}")
print(f"   Recall: {my_class_metrics_improved['recall']}")
print(f"   F1-score: {my_class_metrics_improved['f1']}")

print("ВЫВОДЫ ПО КЛАССИФИКАЦИИ:")

print(f"1. Базовый бейзлайн (sklearn): Accuracy = {class_metrics['accuracy']:.4f}, Macro F1 = {class_metri
print(f"2. Имплементированная модель базового бейзлайна: Accuracy = {my_class_metrics['accuracy']:.4f}, Ma
print(f"3. Улучшенный бейзлайн (sklearn): Accuracy = {class_metrics_improved['accuracy']:.4f}, Macro F1 = 
print(f"4. Имплементированная модель улучшенного бейзлайна: Accuracy = {my_class_metrics_improved['accuracy
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  Accuracy: {((class_metrics_improved['accuracy'] - class_metrics['accuracy']) / class_metrics['ac
print(f"  Macro F1: {((class_metrics_improved['macro_f1'] - class_metrics['macro_f1']) / class_metrics['ma
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в Accuracy: {abs(class_metrics_improved['accuracy'] - my_class_metrics_improved['accuracy
print(f"  Разница в Macro F1: {abs(class_metrics_improved['macro_f1'] - my_class_metrics_improved['macro_f1
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ
                                      Модель  Accuracy  Macro F1
                       Базовый бейзлайн (sklearn)    0.9485  0.953601
   Имплементированная модель базового бейзлайна    0.9485  0.953601
                    Улучшенный бейзлайн (sklearn)    0.9615  0.963740
Имплементированная модель улучшенного бейзлайна    0.9615  0.963740


Детальное сравнение метрик по классам:

1. Базовый бейзлайн (sklearn):
   Precision: [0.97454931 0.97849462 0.91389208]
   Recall: [0.92269076 0.95789474 0.97788698]
   F1-score: [0.94791129 0.96808511 0.94480712]

2. Имплементированная модель базового бейзлайна:
   Precision: [0.97454931 0.97849462 0.91389208]
   Recall: [0.92269076 0.95789474 0.97788698]
   F1-score: [0.94791129 0.96808511 0.94480712]

3. Улучшенный бейзлайн (sklearn):
   Precision: [0.97923157 1.         0.9350057 ]
   Recall: [0.943      0.94117647 0.98795181]
   F1-score: [0.96077433 0.96969697 0.96074985]

4. Имплементированная модель улучшенного бейзлайна:
   Precision: [0.97923157 1.         0.9350057 ]
   Recall: [0.943      0.94117647 0.98795181]
   F1-score: [0.96077433 0.96969697 0.96074985]

ВЫВОДЫ ПО КЛАССИФИКАЦИИ:
1. Базовый бейзлайн (sklearn): Accuracy = 0.9485, Macro F1 = 0.9536
2. Имплементированная модель базового бейзлайна: Accuracy = 0.9485, Macro F1 = 0.9536
3. Улучшенный бейзлайн (sklearn): Accuracy = 0.9615, Macro F1 = 0.9637
4. Имплементированная модель улучшенного бейзлайна: Accuracy = 0.9615, Macro F1 = 0.9637

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  Accuracy: 1.37%
  Macro F1: 1.06%

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в Accuracy: 0.000000
  Разница в Macro F1: 0.000000
```

Регрессия

In [32]:

```python
final_comparison_reg = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'MAE': [
        reg_metrics['mae'],
        my_reg_metrics['mae'],
        reg_metrics_improved['mae'],
        my_reg_metrics_improved['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        my_reg_metrics['rmse'],
        reg_metrics_improved['rmse'],
        my_reg_metrics_improved['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        my_reg_metrics['r2'],
        reg_metrics_improved['r2'],
        my_reg_metrics_improved['r2']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ")
print(final_comparison_reg.to_string(index=False))

print("\nВЫВОДЫ ПО РЕГРЕССИИ:")
print(f"1. Базовый бейзлайн (sklearn): MAE = {reg_metrics['mae']:.4f}, RMSE = {reg_metrics['rmse']:.4f}, R
print(f"2. Имплементированная модель базового бейзлайна: MAE = {my_reg_metrics['mae']:.4f}, RMSE = {my_reg_
print(f"3. Улучшенный бейзлайн (sklearn): MAE = {reg_metrics_improved['mae']:.4f}, RMSE = {reg_metrics_imp
print(f"4. Имплементированная модель улучшенного бейзлайна: MAE = {my_reg_metrics_improved['mae']:.4f}, RMS
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  MAE: {((reg_metrics['mae'] - reg_metrics_improved['mae']) / reg_metrics['mae'] * 100):.2f}% улучш
print(f"  RMSE: {((reg_metrics['rmse'] - reg_metrics_improved['rmse']) / reg_metrics['rmse'] * 100):.2f}% у
print(f"  R²: {((reg_metrics_improved['r2'] - reg_metrics['r2']) / abs(reg_metrics['r2']) * 100):.2f}% улуч
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в MAE: {abs(reg_metrics_improved['mae'] - my_reg_metrics_improved['mae']):.6f}")
print(f"  Разница в RMSE: {abs(reg_metrics_improved['rmse'] - my_reg_metrics_improved['rmse']):.6f}")
print(f"  Разница в R²: {abs(reg_metrics_improved['r2'] - my_reg_metrics_improved['r2']):.6f}")
```

ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ

Модель	MAE	RMSE	R^2
Базовый бейзлайн (sklearn)	1.555981	2.265278	0.525969
Имплементированная модель базового бейзлайна	1.555981	2.265278	0.525969
Улучшенный бейзлайн (sklearn)	1.575222	2.263895	0.526548
Имплементированная модель улучшенного бейзлайна	1.575222	2.263895	0.526548

ВЫВОДЫ ПО РЕГРЕССИИ:
1. Базовый бейзлайн (sklearn): MAE = 1.5560, RMSE = 2.2653, R^2 = 0.5260
2. Имплементированная модель базового бейзлайна: MAE = 1.5560, RMSE = 2.2653, R^2 = 0.5260
3. Улучшенный бейзлайн (sklearn): MAE = 1.5752, RMSE = 2.2639, R^2 = 0.5265
4. Имплементированная модель улучшенного бейзлайна: MAE = 1.5752, RMSE = 2.2639, R^2 = 0.5265

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  MAE: -1.24% улучшение
  RMSE: 0.06% улучшение
  R^2: 0.11% улучшение

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в MAE: 0.000000
  Разница в RMSE: 0.000000
  Разница в R^2: 0.000000

Лабораторная работа №2 (Проведение исследований с логистической и линейной регрессией)

2. Создание бейзлайна и оценка качества

Перейдем к созданию базовых моделей

Классификация

Загрузим датасет и посмотрим на данные

In [1]:
```
import pandas as pd

df_class = pd.read_csv('data/Skyserver_SQL2_27_2018 6_51_39 PM.csv')

print(f"Размерность данных")
print(df_class.shape)
print(f"\nИнформация о данных")
print(df_class.info())
print(f"\nПервые 5 строк")
print(df_class.head())
print(f"\nРаспределение классов")
print(df_class['class'].value_counts())
```

```
Размерность данных
(10000, 18)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   objid      10000 non-null  float64
 1   ra         10000 non-null  float64
 2   dec        10000 non-null  float64
 3   u          10000 non-null  float64
 4   g          10000 non-null  float64
 5   r          10000 non-null  float64
 6   i          10000 non-null  float64
 7   z          10000 non-null  float64
 8   run        10000 non-null  int64
 9   rerun      10000 non-null  int64
 10  camcol     10000 non-null  int64
 11  field      10000 non-null  int64
 12  specobjid  10000 non-null  float64
 13  class      10000 non-null  object
 14  redshift   10000 non-null  float64
 15  plate      10000 non-null  int64
 16  mjd        10000 non-null  int64
 17  fiberid    10000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 1.4+ MB
None

Первые 5 строк
          objid          ra       dec         u         g         r
i  \
0  1.237650e+18  183.531326  0.089693  19.47406  17.04240  15.94699  15.50
342
1  1.237650e+18  183.598370  0.135285  18.66280  17.21449  16.67637  16.48
922
2  1.237650e+18  183.680207  0.126185  19.38298  18.19169  17.47428  17.08
732
3  1.237650e+18  183.870529  0.049911  17.76536  16.60272  16.16116  15.98
233
4  1.237650e+18  183.883288  0.102557  17.55025  16.26342  16.43869  16.55
492

          z  run  rerun  camcol  field     specobjid   class  redshift  pl
ate  \
0  15.22531  752    301       4    267  3.722360e+18    STAR -0.000009   3
306
1  16.39150  752    301       4    267  3.638140e+17    STAR -0.000055
323
2  16.80125  752    301       4    268  3.232740e+17  GALAXY  0.123111
287
3  15.90438  752    301       4    269  3.722370e+18    STAR -0.000111   3
306
4  16.61326  752    301       4    269  3.722370e+18    STAR  0.000590   3
306

     mjd  fiberid
0  54922      491
```

```
1  51615      541
2  52023      513
3  54922      510
4  54922      512

Распределение классов
class
GALAXY    4998
STAR      4152
QSO        850
Name: count, dtype: int64
```

Выделим исходные признаки, которые непосредственно описывают физические свойства объектов и целевую переменную. А также закодируем таргет, так как это категориальный признак.

In [2]:
```
from sklearn.preprocessing import LabelEncoder

X_class = df_class[['u', 'g', 'r', 'i', 'z', 'redshift']]
y_class = df_class['class']

le = LabelEncoder()
y_class_encoded = le.fit_transform(y_class)
class_names = le.classes_
```

Разделим данные на выборку для обучения и тестовую выборку.

In [3]:
```
from sklearn.model_selection import train_test_split

X_train_class, X_test_class, y_train_class, y_test_class = train_test_spl
    X_class, y_class_encoded, test_size=0.2, random_state=42
)
```

Обучим базовую модель логистической регрессии и выполним предсказания на тестовой выборке.

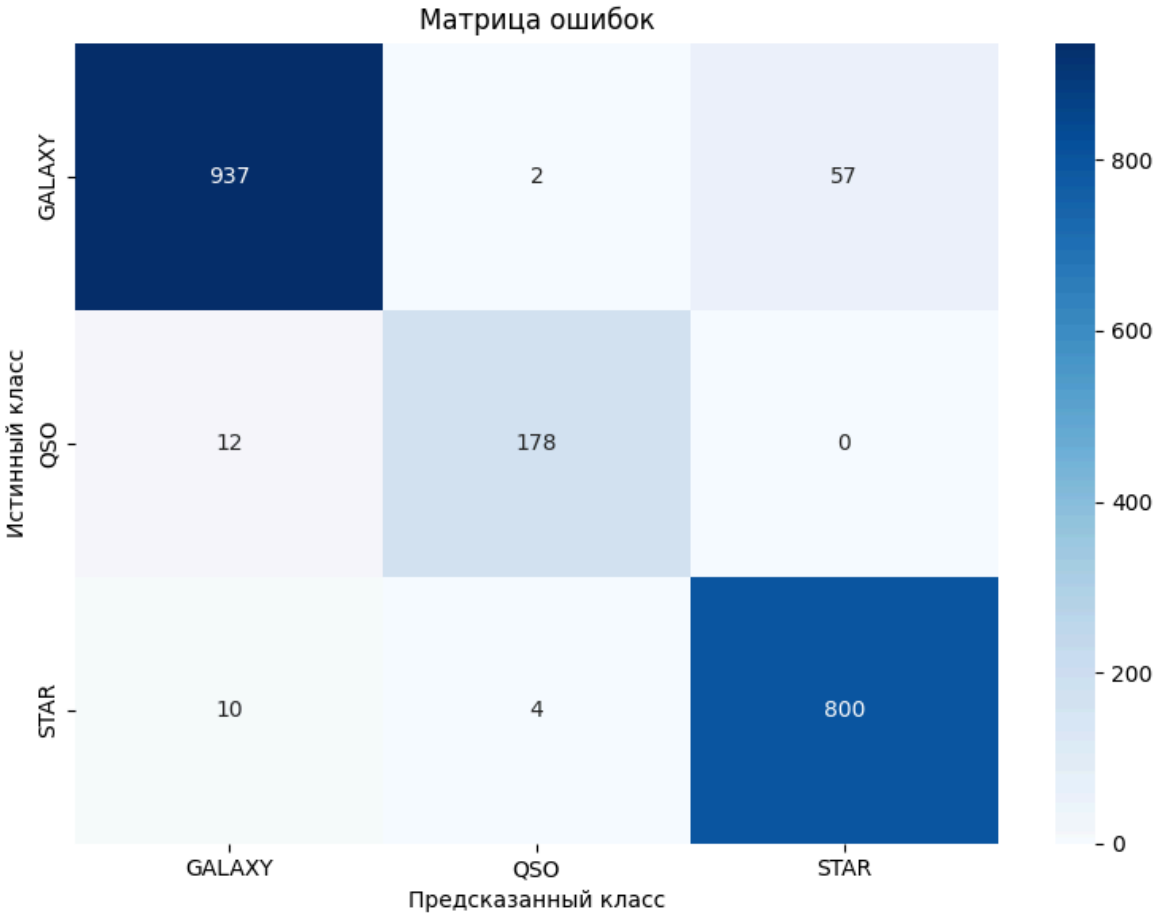
In [4]:
```
from sklearn.linear_model import LogisticRegression

lr_classifier = LogisticRegression(random_state=42, max_iter=1000)
lr_classifier.fit(X_train_class, y_train_class)

y_pred_class = lr_classifier.predict(X_test_class)
```

Опишем функцию, которая будет использоваться для оценки обученной модели классификации.

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix
)

def evaluate_classification_model(y_true, y_pred, class_names):
```

```
    accuracy = accuracy_score(y_true, y_pred)
    print(f"1. Accuracy: {accuracy:.4f}")

    print(f"\n2. Метрики по классам:")
    precision = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)

    metrics_df = pd.DataFrame({
        'Класс': class_names,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
    print(metrics_df.to_string(index=False))

    macro_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"\n3. Macro F1: {macro_f1:.4f}")

    print(f"\n4. Матрица ошибок:")
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
    print(cm_df)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Матрица ошибок')
    plt.ylabel('Истинный класс')
    plt.xlabel('Предсказанный класс')
    plt.tight_layout()
    plt.show()

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'macro_f1': macro_f1,
        'confusion_matrix': cm
    }

class_metrics = evaluate_classification_model(y_test_class, y_pred_class,
```

```
1. Accuracy: 0.9575

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.977059 0.940763  0.958568
   QSO   0.967391 0.936842  0.951872
  STAR   0.933489 0.982801  0.957510

3. Macro F1: 0.9560

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     937    2    57
QSO         12  178     0
STAR        10    4   800
```



Регрессия

Загрузим датасет и посмотрим на данные

In [6]:
```
df_reg = pd.read_csv('data/abalone.csv')

print(f"Размерность данных")
print(df_reg.shape)
print(f"\nИнформация о данных")
print(df_reg.info())
print(f"\nПервые 5 строк")
print(df_reg.head())
print(f"\nСтатистика по числовым признакам")
print(df_reg.describe())
```

```
Размерность данных
(4177, 9)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None

Первые 5 строк
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera wei
ght  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1
010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0
485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1
415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1
140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0
395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7

Статистика по числовым признакам
            Length     Diameter       Height  Whole weight  Shucked weight
\
count  4177.000000  4177.000000  4177.000000   4177.000000     4177.000000
mean      0.523992     0.407881     0.139516      0.828742        0.359367
std       0.120093     0.099240     0.041827      0.490389        0.221963
min       0.075000     0.055000     0.000000      0.002000        0.001000
25%       0.450000     0.350000     0.115000      0.441500        0.186000
50%       0.545000     0.425000     0.140000      0.799500        0.336000
75%       0.615000     0.480000     0.165000      1.153000        0.502000
max       0.815000     0.650000     1.130000      2.825500        1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
```

```
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Выделим признаки и целевую переменную. Закодируем категориальный признак Sex.

In [7]:
```
X_reg = df_reg.drop('Rings', axis=1)
y_reg = df_reg['Rings']

le_sex = LabelEncoder()
X_reg_encoded = X_reg.copy()
X_reg_encoded['Sex'] = le_sex.fit_transform(X_reg['Sex'])
```

Разделим данные на выборку для обучения и тестовую выборку.

In [8]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg_encoded, y_reg, test_size=0.2, random_state=42
)
```

Обучим базовую модель линейной регрессии и выполним предсказания на тестовой выборке.

In [9]:
```
from sklearn.linear_model import LinearRegression

lr_regressor = LinearRegression()
lr_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = lr_regressor.predict(X_test_reg)
```

Опишем функцию, которая будет использоваться для оценки обученной модели регрессии.

In [10]:
```
import numpy as np
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
)

def evaluate_regression_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    print(f"1. MAE: {mae:.4f}")

    mse = mean_squared_error(y_true, y_pred)
    print(f"\n2. MSE: {mse:.4f}")

    rmse = np.sqrt(mse)
    print(f"\n3. RMSE: {rmse:.4f}")

    r2 = r2_score(y_true, y_pred)
    print(f"\n4. R²: {r2:.4f}")

    return {
        'mae': mae,
        'mse': mse,
```

```
        'rmse': rmse,
        'r2': r2
    }

reg_metrics = evaluate_regression_model(y_test_reg, y_pred_reg)
```

```
1. MAE: 1.6306

2. MSE: 5.0625

3. RMSE: 2.2500

4. R²: 0.5323
```

3. Улучшение бейзлайна

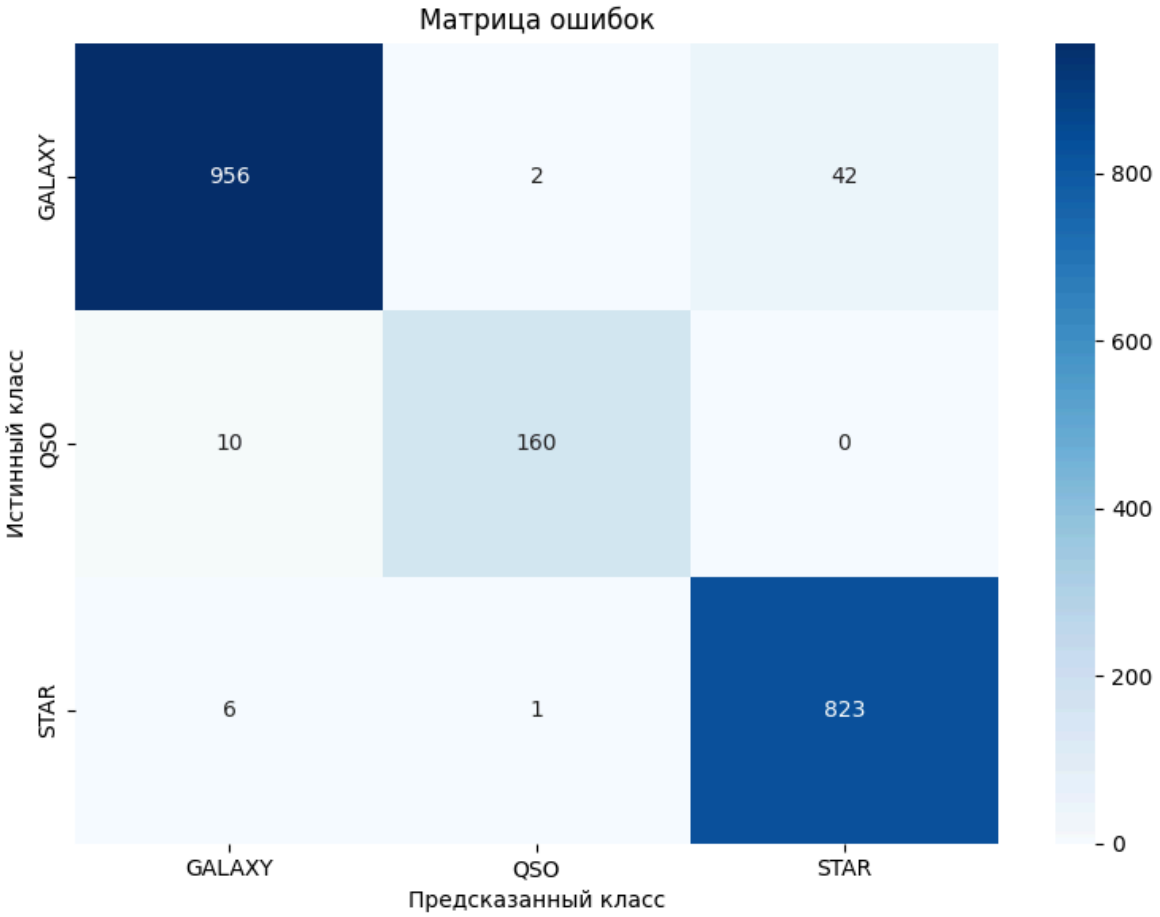
Перейдем к формулированию и проверкам гипотез

Классификация

Гипотеза 1: Добавление стандартизации признаков и стратификации при разбиении на выборки улучшит качество модели.

In [11]:
```
from sklearn.preprocessing import StandardScaler

X_train_class_h1, X_test_class_h1, y_train_class_h1, y_test_class_h1 = tr
    X_class, y_class_encoded, test_size=0.2, stratify=y_class_encoded, ra
)

scaler_class_h1 = StandardScaler()
X_train_class_h1_scaled = scaler_class_h1.fit_transform(X_train_class_h1)
X_test_class_h1_scaled = scaler_class_h1.transform(X_test_class_h1)

lr_classifier_h1 = LogisticRegression(random_state=42, max_iter=1000)
lr_classifier_h1.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_h1 = lr_classifier_h1.predict(X_test_class_h1_scaled)

print("Результаты гипотезы 1:")
class_metrics_h1 = evaluate_classification_model(y_test_class_h1, y_pred_
```

```
Результаты гипотезы 1:
1. Accuracy: 0.9695

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.983539 0.956000  0.969574
   QSO   0.981595 0.941176  0.960961
  STAR   0.951445 0.991566  0.971091

3. Macro F1: 0.9672

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     956    2    42
QSO         10  160     0
STAR         6    1   823
```



Гипотеза 2: Подбор оптимальных гиперпараметров на кросс-валидации улучшит качество модели.

In [12]:
```
from sklearn.model_selection import GridSearchCV

param_grid_h2 = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['lbfgs', 'liblinear', 'saga'],
    'penalty': ['l2']
}

lr_for_tuning = LogisticRegression(random_state=42, max_iter=1000)
grid_search_h2 = GridSearchCV(
    lr_for_tuning,
    param_grid_h2,
    cv=5,
    scoring='f1_macro',
    n_jobs=-1
)
grid_search_h2.fit(X_train_class_h1_scaled, y_train_class_h1)

print("Лучшие параметры:")
for param, value in grid_search_h2.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nЛучший F1-score (кросс-валидация): {grid_search_h2.best_score_:

lr_classifier_h2 = LogisticRegression(
    C=grid_search_h2.best_params_['C'],
    solver=grid_search_h2.best_params_['solver'],
    penalty=grid_search_h2.best_params_['penalty'],
```

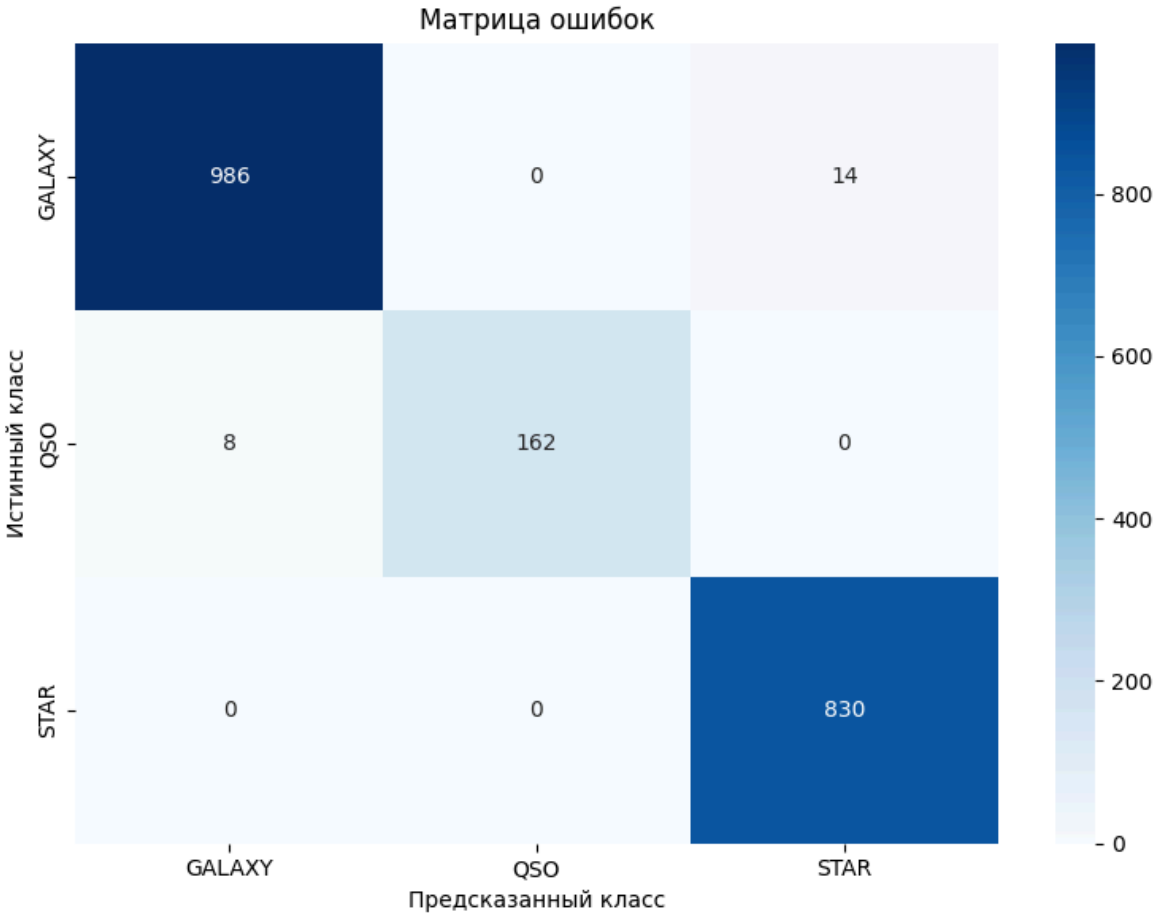
```
    random_state=42,
    max_iter=1000
)
lr_classifier_h2.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_h2 = lr_classifier_h2.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 2:")
class_metrics_h2 = evaluate_classification_model(y_test_class_h1, y_pred_
```

```
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
```

```
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
```

```
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' so
lver for multiclass classification is deprecated. An error will be raised
in 1.8. Either use another solver which supports the multinomial loss or w
rap the estimator in a OneVsRestClassifier to keep applying a one-versus-r
est scheme.
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
```

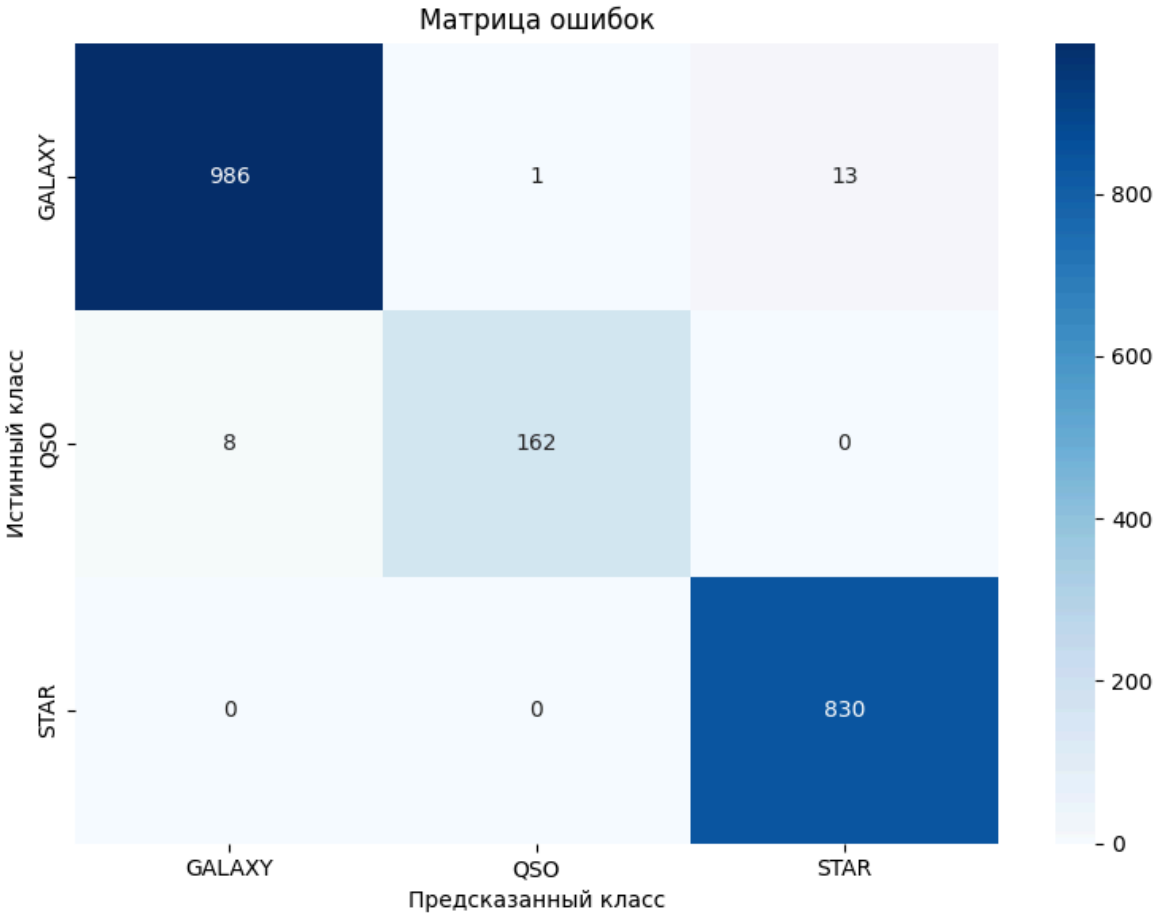
```
rn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
Лучшие параметры:
  C: 100
  penalty: l2
  solver: lbfgs

Лучший F1-score (кросс-валидация): 0.9804

Результаты гипотезы 2:
1. Accuracy: 0.9890

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.991952 0.986000  0.988967
   QSO   1.000000 0.952941  0.975904
  STAR   0.983412 1.000000  0.991637

3. Macro F1: 0.9855

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     986    0    14
QSO          8  162     0
STAR         0    0   830
```



Гипотеза 3: Добавление полиномиальных признаков улучшит качество модели.

In [13]:
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_class_h3_poly = poly.fit_transform(X_train_class_h1_scaled)
X_test_class_h3_poly = poly.transform(X_test_class_h1_scaled)

lr_classifier_h3 = LogisticRegression(
    C=grid_search_h2.best_params_['C'],
    solver=grid_search_h2.best_params_['solver'],
    penalty=grid_search_h2.best_params_['penalty'],
    random_state=42,
    max_iter=1000
)
lr_classifier_h3.fit(X_train_class_h3_poly, y_train_class_h1)
y_pred_class_h3 = lr_classifier_h3.predict(X_test_class_h3_poly)

print("Результаты гипотезы 3:")
class_metrics_h3 = evaluate_classification_model(y_test_class_h1, y_pred_
```

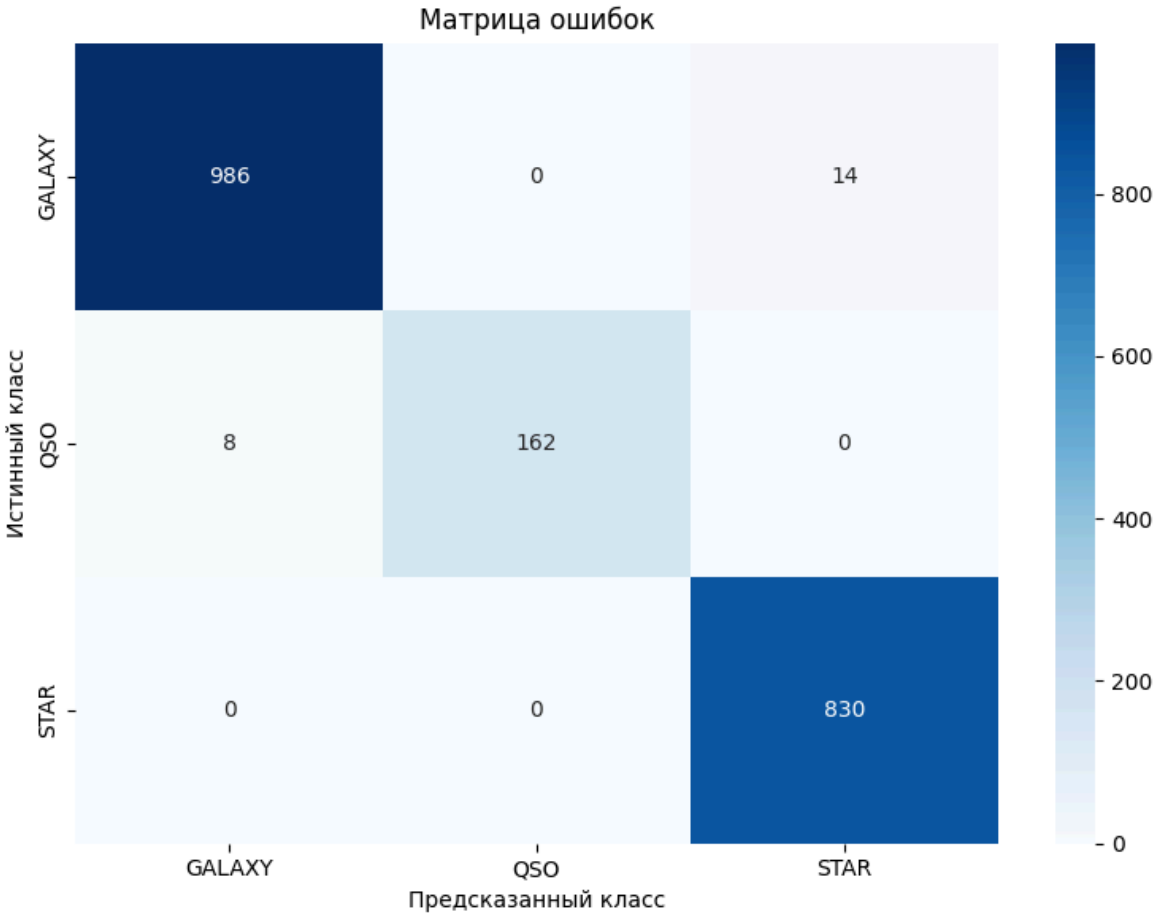
```
Результаты гипотезы 3:
1. Accuracy: 0.9890

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.991952 0.986000  0.988967
   QSO   0.993865 0.952941  0.972973
  STAR   0.984579 1.000000  0.992230

3. Macro F1: 0.9847

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     986    1    13
QSO          8  162     0
STAR         0    0   830
```



Обучим финальную улучшенную модель

In [14]:
```
lr_classifier_improved = LogisticRegression(
    C=grid_search_h2.best_params_['C'],
    solver=grid_search_h2.best_params_['solver'],
    penalty=grid_search_h2.best_params_['penalty'],
    random_state=42,
    max_iter=1000
)
lr_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_improved = lr_classifier_improved.predict(X_test_class_h1_sc

print("Результаты улучшенного бейзлайна для классификации:")
class_metrics_improved = evaluate_classification_model(y_test_class_h1, y
```

```
Результаты улучшенного бейзлайна для классификации:
1. Accuracy: 0.9890

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.991952 0.986000  0.988967
   QSO   1.000000 0.952941  0.975904
  STAR   0.983412 1.000000  0.991637

3. Macro F1: 0.9855

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     986    0    14
QSO          8  162     0
STAR         0    0   830
```



Сравним результаты улучшенного бейзлайна с базовым бейзлайном

In [15]:
```
comparison_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'Accuracy': [class_metrics['accuracy'], class_metrics_improved['accur
    'Macro F1': [class_metrics['macro_f1'], class_metrics_improved['macro
})

print("Сравнение базового и улучшенного бейзлайна для классификации:")
print(comparison_class.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для классификации:
            Модель  Accuracy  Macro F1
   Базовый бейзлайн    0.9575  0.955983
Улучшенный бейзлайн    0.9890  0.985502
```

Регрессия

Гипотеза 1: Добавление стандартизации признаков улучшит качество модели.

In [16]:
```
scaler_reg_h1 = StandardScaler()
X_train_reg_h1_scaled = scaler_reg_h1.fit_transform(X_train_reg)
X_test_reg_h1_scaled = scaler_reg_h1.transform(X_test_reg)

lr_regressor_h1 = LinearRegression()
lr_regressor_h1.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_h1 = lr_regressor_h1.predict(X_test_reg_h1_scaled)

print("Результаты гипотезы 1:")
reg_metrics_h1 = evaluate_regression_model(y_test_reg, y_pred_reg_h1)
```

```
Результаты гипотезы 1:
1. MAE: 1.6306

2. MSE: 5.0625

3. RMSE: 2.2500

4. R²: 0.5323
```

Гипотеза 2: Использование Ridge регрессии с подбором параметра регуляризации улучшит качество модели.

In [17]:
```
from sklearn.linear_model import Ridge

param_grid_reg_h2 = {
    'alpha': [0.01, 0.1, 1, 10, 100, 1000]
}

ridge_for_tuning = Ridge(random_state=42)
grid_search_reg_h2 = GridSearchCV(
    ridge_for_tuning,
    param_grid_reg_h2,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search_reg_h2.fit(X_train_reg_h1_scaled, y_train_reg)

print("Лучшие параметры:")
for param, value in grid_search_reg_h2.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nЛучший MSE (кросс-валидация): {-grid_search_reg_h2.best_score_:

ridge_regressor_h2 = Ridge(
    alpha=grid_search_reg_h2.best_params_['alpha'],
    random_state=42
)
ridge_regressor_h2.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_h2 = ridge_regressor_h2.predict(X_test_reg_h1_scaled)
```

```
print("\nРезультаты гипотезы 2:")
reg_metrics_h2 = evaluate_regression_model(y_test_reg, y_pred_reg_h2)
```

```
Лучшие параметры:
  alpha: 0.1

Лучший MSE (кросс-валидация): 4.9880

Результаты гипотезы 2:
1. MAE: 1.6306

2. MSE: 5.0625

3. RMSE: 2.2500

4. R²: 0.5323
```

Гипотеза 3: Добавление полиномиальных признаков улучшит качество модели.

In [18]:
```
poly_reg = PolynomialFeatures(degree=2, include_bias=False)
X_train_reg_h3_poly = poly_reg.fit_transform(X_train_reg_h1_scaled)
X_test_reg_h3_poly = poly_reg.transform(X_test_reg_h1_scaled)

ridge_regressor_h3 = Ridge(
    alpha=grid_search_reg_h2.best_params_['alpha'],
    random_state=42
)
ridge_regressor_h3.fit(X_train_reg_h3_poly, y_train_reg)
y_pred_reg_h3 = ridge_regressor_h3.predict(X_test_reg_h3_poly)

print("Результаты гипотезы 3:")
reg_metrics_h3 = evaluate_regression_model(y_test_reg, y_pred_reg_h3)
```

```
Результаты гипотезы 3:
1. MAE: 1.5493

2. MSE: 4.8623

3. RMSE: 2.2051

4. R²: 0.5508
```

Обучим финальную улучшенную модель

In [19]:
```
ridge_regressor_improved = Ridge(
    alpha=grid_search_reg_h2.best_params_['alpha'],
    random_state=42
)
ridge_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_improved = ridge_regressor_improved.predict(X_test_reg_h1_scal

print("Результаты улучшенного бейзлайна для регрессии:")
reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_reg_i
```

```
Результаты улучшенного бейзлайна для регрессии:
1. MAE: 1.6306

2. MSE: 5.0625

3. RMSE: 2.2500

4. R²: 0.5323
```

Сравним результаты улучшенного бейзлайна с базовым бейзлайном

In [20]:
```python
comparison_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'MAE': [reg_metrics['mae'], reg_metrics_improved['mae']],
    'RMSE': [reg_metrics['rmse'], reg_metrics_improved['rmse']],
    'R²': [reg_metrics['r2'], reg_metrics_improved['r2']]
})

print("Сравнение базового и улучшенного бейзлайна для регрессии:")
print(comparison_reg.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для регрессии:
            Модель      MAE     RMSE       R²
   Базовый бейзлайн 1.630561 2.250008 0.532338
Улучшенный бейзлайн 1.630633 2.250000 0.532342
```

4. Имплементация алгоритма машинного обучения

Перейдем к имлпементации алгоритмов

Классификация

Реализуем алгоритм логистической регрессии для классификации

In [21]:
```python
class MyLogisticRegression:
    def __init__(self, learning_rate=0.01, max_iter=1000, C=1.0, random_s
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.C = C
        self.random_state = random_state
        self.weights = None
        self.bias = None
        self.classes_ = None
        self.n_classes_ = None

    def _sigmoid(self, z):
        z = np.clip(z, -500, 500)
        return 1 / (1 + np.exp(-z))

    def _softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
```

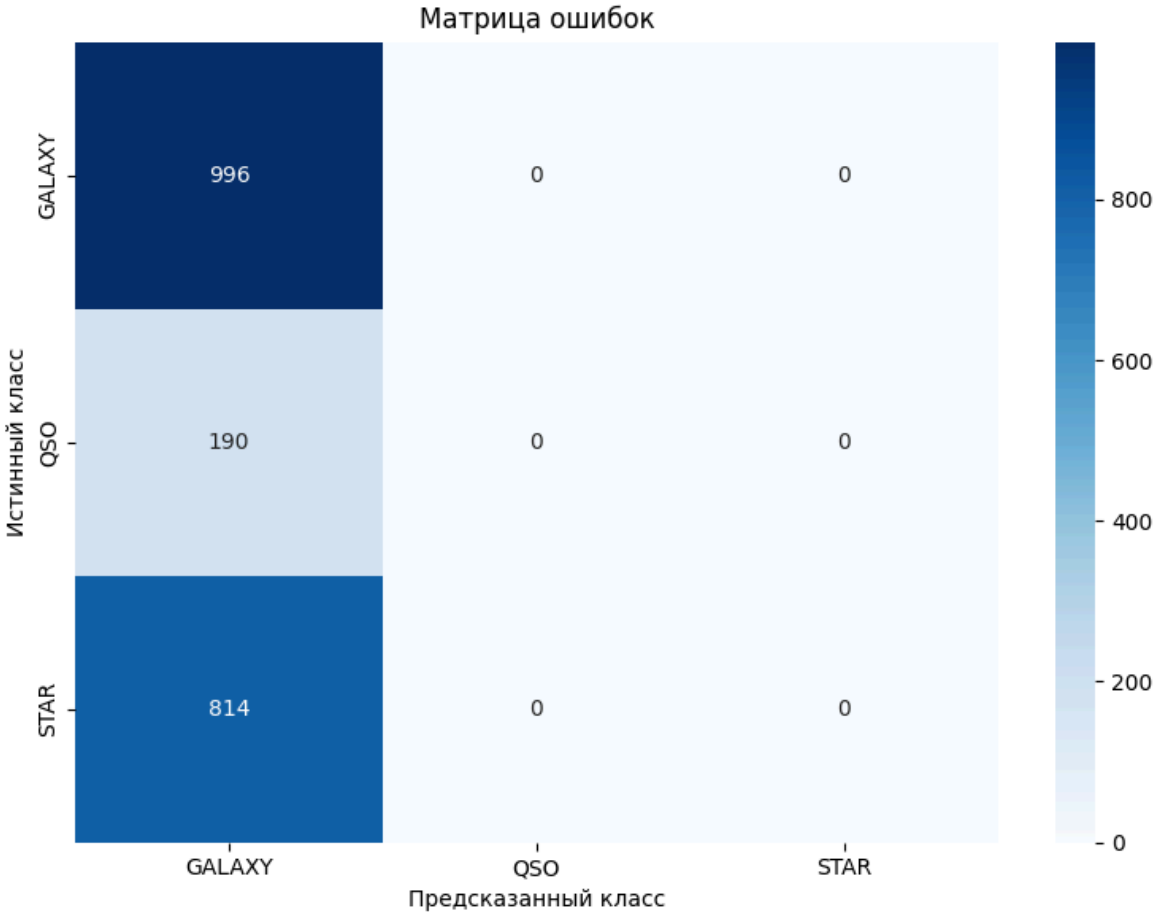
```
        if self.random_state is not None:
            np.random.seed(self.random_state)

        self.classes_ = np.unique(y)
        self.n_classes_ = len(self.classes_)

        n_samples, n_features = X.shape

        if self.n_classes_ == 2:
            y_binary = (y == self.classes_[1]).astype(float)
            self.weights = np.random.randn(n_features) * 0.01
            self.bias = 0.0

            for i in range(self.max_iter):
                z = np.dot(X, self.weights) + self.bias
                predictions = self._sigmoid(z)

                dw = (1 / n_samples) * np.dot(X.T, (predictions - y_binar
                db = (1 / n_samples) * np.sum(predictions - y_binary)

                self.weights -= self.learning_rate * dw
                self.bias -= self.learning_rate * db
        else:
            self.weights = np.random.randn(self.n_classes_, n_features) *
            self.bias = np.zeros(self.n_classes_)

            for i in range(self.max_iter):
                z = np.dot(X, self.weights.T) + self.bias
                predictions = self._softmax(z)

                y_one_hot = np.zeros((n_samples, self.n_classes_))
                for idx, cls in enumerate(self.classes_):
                    y_one_hot[:, idx] = (y == cls).astype(float)

                error = predictions - y_one_hot
                dw = (1 / n_samples) * np.dot(error.T, X) + (1 / self.C)
                db = (1 / n_samples) * np.sum(error, axis=0)

                self.weights -= self.learning_rate * dw
                self.bias -= self.learning_rate * db

        return self

    def predict_proba(self, X):
        X = np.array(X)

        if self.n_classes_ == 2:
            z = np.dot(X, self.weights) + self.bias
            proba = self._sigmoid(z)
            return np.column_stack([1 - proba, proba])
        else:
            z = np.dot(X, self.weights.T) + self.bias
            return self._softmax(z)

    def predict(self, X):
        proba = self.predict_proba(X)
        return self.classes_[np.argmax(proba, axis=1)]
```

Обучим имплементированную модель на исходных данных

In [22]:
```
my_lr_classifier = MyLogisticRegression(learning_rate=0.01, max_iter=1000
my_lr_classifier.fit(X_train_class.values, y_train_class)
y_pred_my_class = my_lr_classifier.predict(X_test_class.values)

print("Результаты имплементированной модели классификации:")
my_class_metrics = evaluate_classification_model(y_test_class, y_pred_my_
```

```
Результаты имплементированной модели классификации:
1. Accuracy: 0.4980

2. Метрики по классам:
 Класс  Precision  Recall  F1-score
GALAXY      0.498     1.0  0.664887
   QSO      0.000     0.0  0.000000
  STAR      0.000     0.0  0.000000

3. Macro F1: 0.2216

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     996    0     0
QSO        190    0     0
STAR       814    0     0
```

```
/home/zendroix/labs/AI_Frameworks/.venv/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1731: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `
zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape
[0])
```



Сравним результаты имплементированной модели с базовым бейзлайном

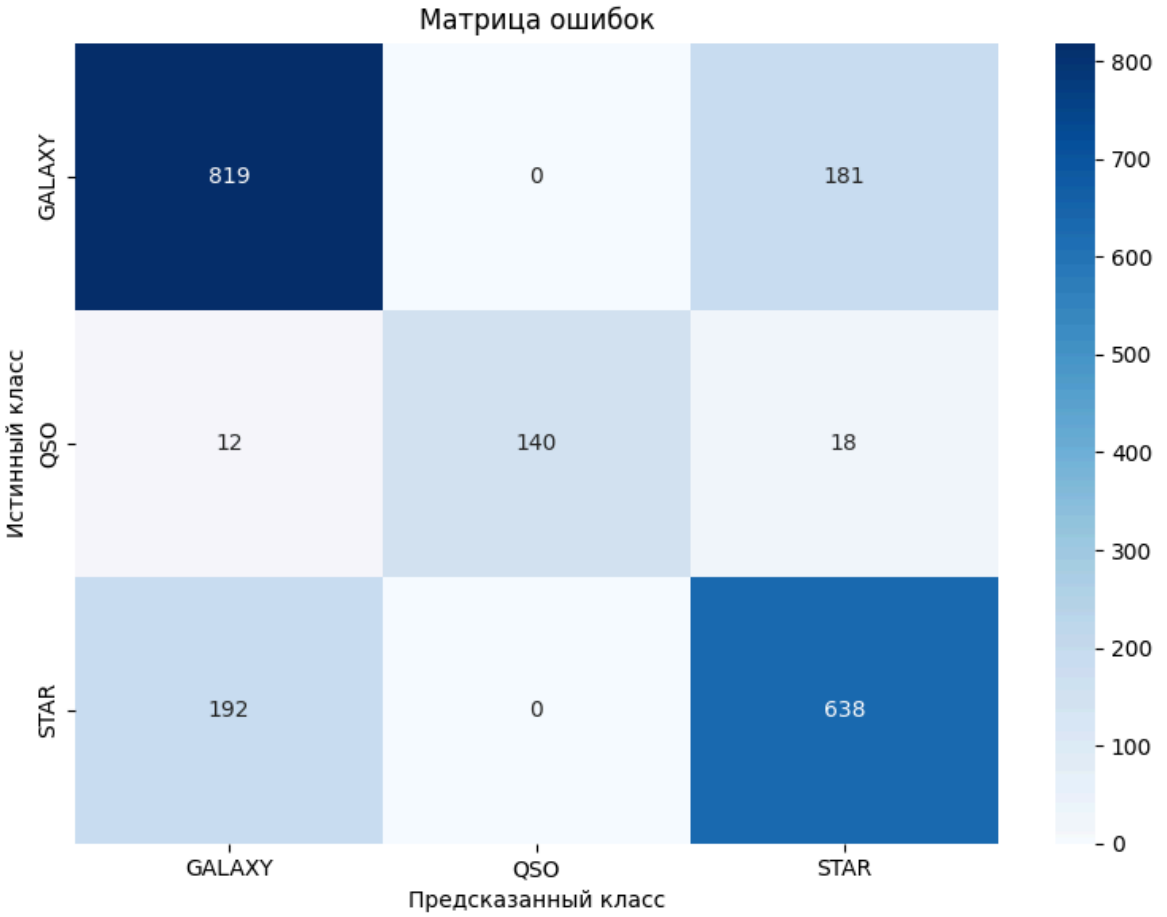
In [23]:
```
comparison_my_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель']
    'Accuracy': [class_metrics['accuracy'], my_class_metrics['accuracy']]
    'Macro F1': [class_metrics['macro_f1'], my_class_metrics['macro_f1']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_class.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                  Модель  Accuracy  Macro F1
Базовый бейзлайн (sklearn)    0.9575  0.955983
 Имплементированная модель    0.4980  0.221629
```

Теперь применим техники из улучшенного бейзлайна

In [24]:
```
best_C = grid_search_h2.best_params_['C']

my_lr_classifier_improved = MyLogisticRegression(
    learning_rate=0.01,
    max_iter=1000,
    C=best_C,
    random_state=42
)
my_lr_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_my_class_improved = my_lr_classifier_improved.predict(X_test_class

print("Результаты имплементированной модели с улучшениями:")
my_class_metrics_improved = evaluate_classification_model(y_test_class_h1
```

```
Результаты имплементированной модели с улучшениями:
1. Accuracy: 0.7985

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.800587 0.819000  0.809689
   QSO   1.000000 0.823529  0.903226
  STAR   0.762246 0.768675  0.765447

3. Macro F1: 0.8261

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     819    0   181
QSO         12  140    18
STAR       192    0   638
```



Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [25]:
```
comparison_my_class_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модел
    'Accuracy': [class_metrics_improved['accuracy'], my_class_metrics_imp
    'Macro F1': [class_metrics_improved['macro_f1'], my_class_metrics_imp
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бей
print(comparison_my_class_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                                Модель  Accuracy  Macro F1
          Улучшенный бейзлайн (sklearn)    0.9890  0.985502
Имплементированная модель с улучшениями    0.7985  0.826120
```

Регрессия

Реализуем алгоритм линейной регрессии

In [26]:
```
class MyLinearRegression:
    def __init__(self, learning_rate=0.01, max_iter=1000, alpha=0.0, rand
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.alpha = alpha
        self.random_state = random_state
        self.weights = None
        self.bias = None
```

```
    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)

        if self.random_state is not None:
            np.random.seed(self.random_state)

        n_samples, n_features = X.shape

        self.weights = np.random.randn(n_features) * 0.01
        self.bias = 0.0

        for i in range(self.max_iter):
            predictions = np.dot(X, self.weights) + self.bias

            dw = (1 / n_samples) * np.dot(X.T, (predictions - y)) + (self
            db = (1 / n_samples) * np.sum(predictions - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

        return self

    def predict(self, X):
        X = np.array(X)
        return np.dot(X, self.weights) + self.bias
```

Обучим имплементированную модель на исходных данных

In [27]:
```
my_lr_regressor = MyLinearRegression(learning_rate=0.01, max_iter=1000, a
my_lr_regressor.fit(X_train_reg.values, y_train_reg.values)
y_pred_my_reg = my_lr_regressor.predict(X_test_reg.values)

print("Результаты имплементированной модели регрессии:")
my_reg_metrics = evaluate_regression_model(y_test_reg, y_pred_my_reg)
```

```
Результаты имплементированной модели регрессии:
1. MAE: 1.9437

2. MSE: 7.2421

3. RMSE: 2.6911

4. R²: 0.3310
```

Сравним результаты имплементированной модели с базовым бейзлайном

In [28]:
```
comparison_my_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель']
    'MAE': [reg_metrics['mae'], my_reg_metrics['mae']],
    'RMSE': [reg_metrics['rmse'], my_reg_metrics['rmse']],
    'R²': [reg_metrics['r2'], my_reg_metrics['r2']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_reg.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                    Модель      MAE     RMSE       R²
Базовый бейзлайн (sklearn) 1.630561 2.250008 0.532338
 Имплементированная модель 1.943738 2.691109 0.331000
```

Теперь применим техники из улучшенного бейзлайна

In [29]:
```
best_alpha = grid_search_reg_h2.best_params_['alpha']

my_lr_regressor_improved = MyLinearRegression(
    learning_rate=0.01,
    max_iter=1000,
    alpha=best_alpha,
    random_state=42
)
my_lr_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg.values)
y_pred_my_reg_improved = my_lr_regressor_improved.predict(X_test_reg_h1_s

print("Результаты имплементированной модели с улучшениями:")
my_reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_my
```

```
Результаты имплементированной модели с улучшениями:
1. MAE: 1.6754

2. MSE: 5.4186

3. RMSE: 2.3278

4. R²: 0.4994
```

Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [30]:
```
comparison_my_reg_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модел
    'MAE': [reg_metrics_improved['mae'], my_reg_metrics_improved['mae']],
    'RMSE': [reg_metrics_improved['rmse'], my_reg_metrics_improved['rmse'
    'R²': [reg_metrics_improved['r2'], my_reg_metrics_improved['r2']]
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бей
print(comparison_my_reg_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                              Модель      MAE     RMSE       R²
          Улучшенный бейзлайн (sklearn) 1.630633 2.250000 0.532342
Имплементированная модель с улучшениями 1.675446 2.327788 0.499447
```

Общие выводы по результатам всех моделей

Сравним все 4 модели для классификации и регрессии: базовый бейзлайн из sklearn, имплементированную модель базового бейзлайна, модель с улучшенным бейзлайном из sklearn и имплементированную модель улучшенного бейзлайна.

Классификация

In [31]:
```
final_comparison_class = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'Accuracy': [
        class_metrics['accuracy'],
        my_class_metrics['accuracy'],
        class_metrics_improved['accuracy'],
        my_class_metrics_improved['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        my_class_metrics['macro_f1'],
        class_metrics_improved['macro_f1'],
        my_class_metrics_improved['macro_f1']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ")
print(final_comparison_class.to_string(index=False))
print("\nВЫВОДЫ ПО КЛАССИФИКАЦИИ:")
print(f"1. Базовый бейзлайн (sklearn): Accuracy = {class_metrics['accurac
print(f"2. Имплементированная модель базового бейзлайна: Accuracy = {my_c
print(f"3. Улучшенный бейзлайн (sklearn): Accuracy = {class_metrics_impro
print(f"4. Имплементированная модель улучшенного бейзлайна: Accuracy = {m
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sk
print(f"  Accuracy: {((class_metrics_improved['accuracy'] - class_metrics
print(f"  Macro F1: {((class_metrics_improved['macro_f1'] - class_metrics
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзла
print(f"  Разница в Accuracy: {abs(class_metrics_improved['accuracy'] - m
print(f"  Разница в Macro F1: {abs(class_metrics_improved['macro_f1'] - m
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ
                                    Модель  Accuracy  Macro F1
                  Базовый бейзлайн (sklearn)    0.9575  0.955983
 Имплементированная модель базового бейзлайна    0.4980  0.221629
                 Улучшенный бейзлайн (sklearn)    0.9890  0.985502
Имплементированная модель улучшенного бейзлайна    0.7985  0.826120

ВЫВОДЫ ПО КЛАССИФИКАЦИИ:
1. Базовый бейзлайн (sklearn): Accuracy = 0.9575, Macro F1 = 0.9560
2. Имплементированная модель базового бейзлайна: Accuracy = 0.4980, Macro
F1 = 0.2216
3. Улучшенный бейзлайн (sklearn): Accuracy = 0.9890, Macro F1 = 0.9855
4. Имплементированная модель улучшенного бейзлайна: Accuracy = 0.7985, Mac
ro F1 = 0.8261

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  Accuracy: 3.29%
  Macro F1: 3.09%

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в Accuracy: 0.190500
  Разница в Macro F1: 0.159382
```

Регрессия

In [32]:
```
final_comparison_reg = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'MAE': [
        reg_metrics['mae'],
        my_reg_metrics['mae'],
        reg_metrics_improved['mae'],
        my_reg_metrics_improved['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        my_reg_metrics['rmse'],
        reg_metrics_improved['rmse'],
        my_reg_metrics_improved['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        my_reg_metrics['r2'],
        reg_metrics_improved['r2'],
        my_reg_metrics_improved['r2']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ")
print(final_comparison_reg.to_string(index=False))
print("\nВЫВОДЫ ПО РЕГРЕССИИ:")
print(f"1. Базовый бейзлайн (sklearn): MAE = {reg_metrics['mae']:.4f}, RM
print(f"2. Имплементированная модель базового бейзлайна: MAE = {my_reg_me
print(f"3. Улучшенный бейзлайн (sklearn): MAE = {reg_metrics_improved['ma
print(f"4. Имплементированная модель улучшенного бейзлайна: MAE = {my_reg
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sk
print(f"  MAE: {((reg_metrics['mae'] - reg_metrics_improved['mae']) / reg
print(f"  RMSE: {((reg_metrics['rmse'] - reg_metrics_improved['rmse']) /
print(f"  R²: {((reg_metrics_improved['r2'] - reg_metrics['r2']) / abs(re
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзла
print(f"  Разница в MAE: {abs(reg_metrics_improved['mae'] - my_reg_metric
print(f"  Разница в RMSE: {abs(reg_metrics_improved['rmse'] - my_reg_metr
print(f"  Разница в R²: {abs(reg_metrics_improved['r2'] - my_reg_metrics_
```

ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ

Модель	MAE	RMSE	R^2
Базовый бейзлайн (sklearn)	1.630561	2.250008	0.532338
Имплементированная модель базового бейзлайна	1.943738	2.691109	0.331000
Улучшенный бейзлайн (sklearn)	1.630633	2.250000	0.532342
Имплементированная модель улучшенного бейзлайна	1.675446	2.327788	0.499447

ВЫВОДЫ ПО РЕГРЕССИИ:
1. Базовый бейзлайн (sklearn): MAE = 1.6306, RMSE = 2.2500, R^2 = 0.5323
2. Имплементированная модель базового бейзлайна: MAE = 1.9437, RMSE = 2.69
11, R^2 = 0.3310
3. Улучшенный бейзлайн (sklearn): MAE = 1.6306, RMSE = 2.2500, R^2 = 0.5323
4. Имплементированная модель улучшенного бейзлайна: MAE = 1.6754, RMSE =
2.3278, R^2 = 0.4994

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  MAE: -0.00% улучшение
  RMSE: 0.00% улучшение
  R^2: 0.00% улучшение

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в MAE: 0.044813
  Разница в RMSE: 0.077788
  Разница в R^2: 0.032895

Лабораторная работа №3 (Проведение исследований с решающим деревом)

2. Создание бейзлайна и оценка качества

Перейдем к созданию базовых моделей

Классификация

Загрузим датасет и посмотрим на данные

In [1]:
```
import pandas as pd

df_class = pd.read_csv('data/Skyserver_SQL2_27_2018 6_51_39 PM.csv')

print(f"Размерность данных")
print(df_class.shape)
print(f"\nИнформация о данных")
print(df_class.info())
print(f"\nПервые 5 строк")
print(df_class.head())
print(f"\nРаспределение классов")
print(df_class['class'].value_counts())
```

```
Размерность данных
(10000, 18)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   objid      10000 non-null  float64
 1   ra         10000 non-null  float64
 2   dec        10000 non-null  float64
 3   u          10000 non-null  float64
 4   g          10000 non-null  float64
 5   r          10000 non-null  float64
 6   i          10000 non-null  float64
 7   z          10000 non-null  float64
 8   run        10000 non-null  int64
 9   rerun      10000 non-null  int64
 10  camcol     10000 non-null  int64
 11  field      10000 non-null  int64
 12  specobjid  10000 non-null  float64
 13  class      10000 non-null  object
 14  redshift   10000 non-null  float64
 15  plate      10000 non-null  int64
 16  mjd        10000 non-null  int64
 17  fiberid    10000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 1.4+ MB
None

Первые 5 строк
          objid          ra       dec         u         g         r         i  \
0  1.237650e+18  183.531326  0.089693  19.47406  17.04240  15.94699  15.50342
1  1.237650e+18  183.598370  0.135285  18.66280  17.21449  16.67637  16.48922
2  1.237650e+18  183.680207  0.126185  19.38298  18.19169  17.47428  17.08732
3  1.237650e+18  183.870529  0.049911  17.76536  16.60272  16.16116  15.98233
4  1.237650e+18  183.883288  0.102557  17.55025  16.26342  16.43869  16.55492

          z  run  rerun  camcol  field     specobjid   class  redshift  plate  \
0  15.22531  752    301       4    267  3.722360e+18    STAR -0.000009   3306
1  16.39150  752    301       4    267  3.638140e+17    STAR -0.000055    323
2  16.80125  752    301       4    268  3.232740e+17  GALAXY  0.123111    287
3  15.90438  752    301       4    269  3.722370e+18    STAR -0.000111   3306
4  16.61326  752    301       4    269  3.722370e+18    STAR  0.000590   3306

     mjd  fiberid
0  54922      491
1  51615      541
2  52023      513
3  54922      510
4  54922      512

Распределение классов
class
GALAXY    4998
STAR      4152
QSO        850
Name: count, dtype: int64
```

Выделим исходные признаки, которые непосредственно описывают физические свойства объектов и целевую переменную. А также закодируем таргет, так как это категориальный признак.

In [2]:
```python
from sklearn.preprocessing import LabelEncoder

X_class = df_class[['u', 'g', 'r', 'i', 'z', 'redshift']]
y_class = df_class['class']

le = LabelEncoder()
y_class_encoded = le.fit_transform(y_class)
class_names = le.classes_
```

Разделим данные на выборку для обучения и тестовую выборку.

In [3]:
```python
from sklearn.model_selection import train_test_split

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class_encoded, test_size=0.2, random_state=42
)
```

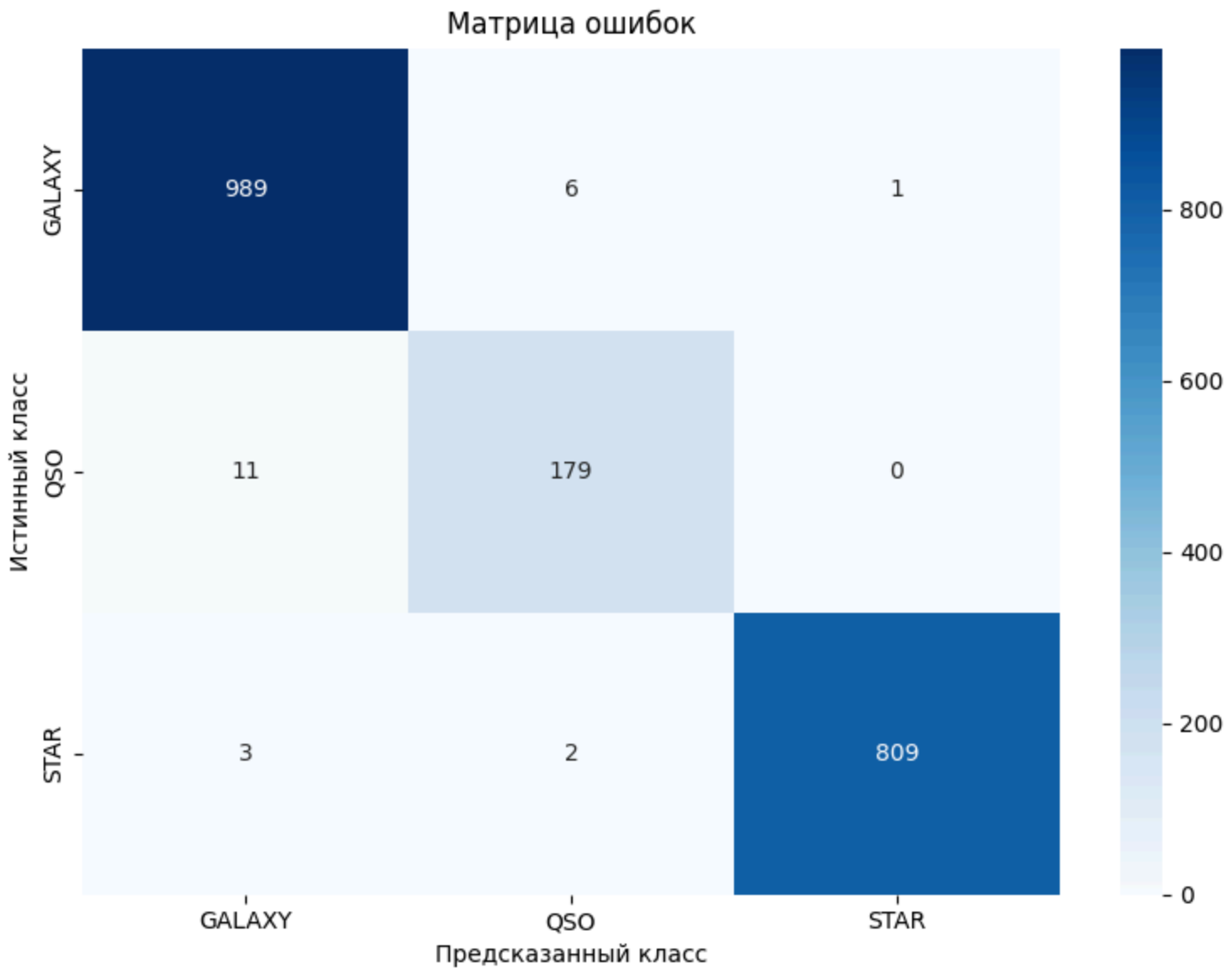
Обучим базовую модель DecisionTree и выполним предсказания на тестовой выборке.

In [4]:
```python
from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_class, y_train_class)

y_pred_class = dt_classifier.predict(X_test_class)
```

Опишем функцию, которая будет использоваться для оценки обученной модели классификации.

In [5]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix
)

def evaluate_classification_model(y_true, y_pred, class_names):
    accuracy = accuracy_score(y_true, y_pred)
    print(f"1. Accuracy: {accuracy:.4f}")

    print(f"\n2. Метрики по классам:")
    precision = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)

    metrics_df = pd.DataFrame({
        'Класс': class_names,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
    print(metrics_df.to_string(index=False))

    macro_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"\n3. Macro F1: {macro_f1:.4f}")

    print(f"\n4. Матрица ошибок:")
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
    print(cm_df)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Матрица ошибок')
    plt.ylabel('Истинный класс')
    plt.xlabel('Предсказанный класс')
    plt.tight_layout()
    plt.show()

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'macro_f1': macro_f1,
        'confusion_matrix': cm
    }

class_metrics = evaluate_classification_model(y_test_class, y_pred_class, class_names)
```

```
1. Accuracy: 0.9885

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.986042 0.992972  0.989495
   QSO   0.957219 0.942105  0.949602
  STAR   0.998765 0.993857  0.996305

3. Macro F1: 0.9785

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     989    6     1
QSO         11  179     0
STAR         3    2   809
```



Регрессия

Загрузим датасет и посмотрим на данные

In [6]:
```
df_reg = pd.read_csv('data/abalone.csv')

print(f"Размерность данных")
print(df_reg.shape)
print(f"\nИнформация о данных")
print(df_reg.info())
print(f"\nПервые 5 строк")
print(df_reg.head())
print(f"\nСтатистика по числовым признакам")
print(df_reg.describe())
```

```
Размерность данных
(4177, 9)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None

Первые 5 строк
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7

Статистика по числовым признакам
            Length     Diameter       Height  Whole weight  Shucked weight  \
count  4177.000000  4177.000000  4177.000000   4177.000000     4177.000000
mean      0.523992     0.407881     0.139516      0.828742        0.359367
std       0.120093     0.099240     0.041827      0.490389        0.221963
min       0.075000     0.055000     0.000000      0.002000        0.001000
25%       0.450000     0.350000     0.115000      0.441500        0.186000
50%       0.545000     0.425000     0.140000      0.799500        0.336000
75%       0.615000     0.480000     0.165000      1.153000        0.502000
max       0.815000     0.650000     1.130000      2.825500        1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Выделим признаки и целевую переменную. Закодируем категориальный признак Sex.

In [7]:
```
X_reg = df_reg.drop('Rings', axis=1)
y_reg = df_reg['Rings']

le_sex = LabelEncoder()
X_reg_encoded = X_reg.copy()
X_reg_encoded['Sex'] = le_sex.fit_transform(X_reg['Sex'])
```

Разделим данные на выборку для обучения и тестовую выборку.

In [8]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg_encoded, y_reg, test_size=0.2, random_state=42
)
```

Обучим базовую модель DecisionTree и выполним предсказания на тестовой выборке.

In [9]:
```
from sklearn.tree import DecisionTreeRegressor

dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = dt_regressor.predict(X_test_reg)
```

Опишем функцию, которая будет использоваться для оценки обученной модели регрессии.

In [10]:
```
import numpy as np
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
```

```
    r2_score,
)

def evaluate_regression_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    print(f"1. MAE: {mae:.4f}")

    mse = mean_squared_error(y_true, y_pred)
    print(f"\n2. MSE: {mse:.4f}")

    rmse = np.sqrt(mse)
    print(f"\n3. RMSE: {rmse:.4f}")

    r2 = r2_score(y_true, y_pred)
    print(f"\n4. R²: {r2:.4f}")

    return {
        'mae': mae,
        'mse': mse,
        'rmse': rmse,
        'r2': r2
    }

reg_metrics = evaluate_regression_model(y_test_reg, y_pred_reg)
```

```
1. MAE: 2.1782

2. MSE: 9.8194

3. RMSE: 3.1336

4. R²: 0.0929
```

3. Улучшение бейзлайна

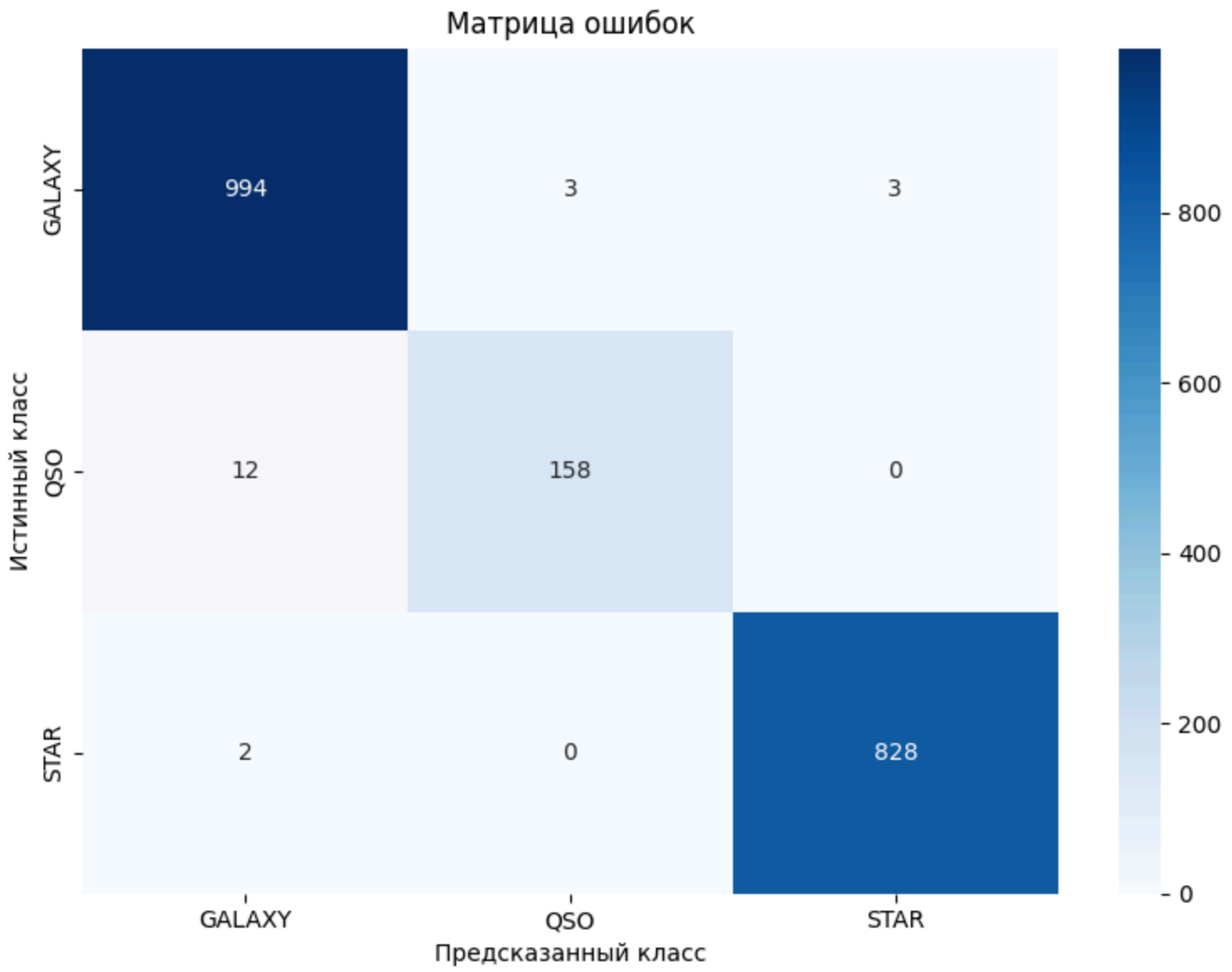
Перейдем к формулированию и проверкам гипотез

Классификация

Гипотеза 1: Добавление стандартизации признаков и стратификации при разбиении на выборки улучшит качество модели.

In [11]:
```
from sklearn.preprocessing import StandardScaler

X_train_class_h1, X_test_class_h1, y_train_class_h1, y_test_class_h1 = train_test_split(
    X_class, y_class_encoded, test_size=0.2, stratify=y_class_encoded, random_state=42
)

scaler_class_h1 = StandardScaler()
X_train_class_h1_scaled = scaler_class_h1.fit_transform(X_train_class_h1)
X_test_class_h1_scaled = scaler_class_h1.transform(X_test_class_h1)

dt_classifier_h1 = DecisionTreeClassifier(random_state=42)
dt_classifier_h1.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_h1 = dt_classifier_h1.predict(X_test_class_h1_scaled)

print("Результаты гипотезы 1:")
class_metrics_h1 = evaluate_classification_model(y_test_class_h1, y_pred_class_h1, class_names)
```

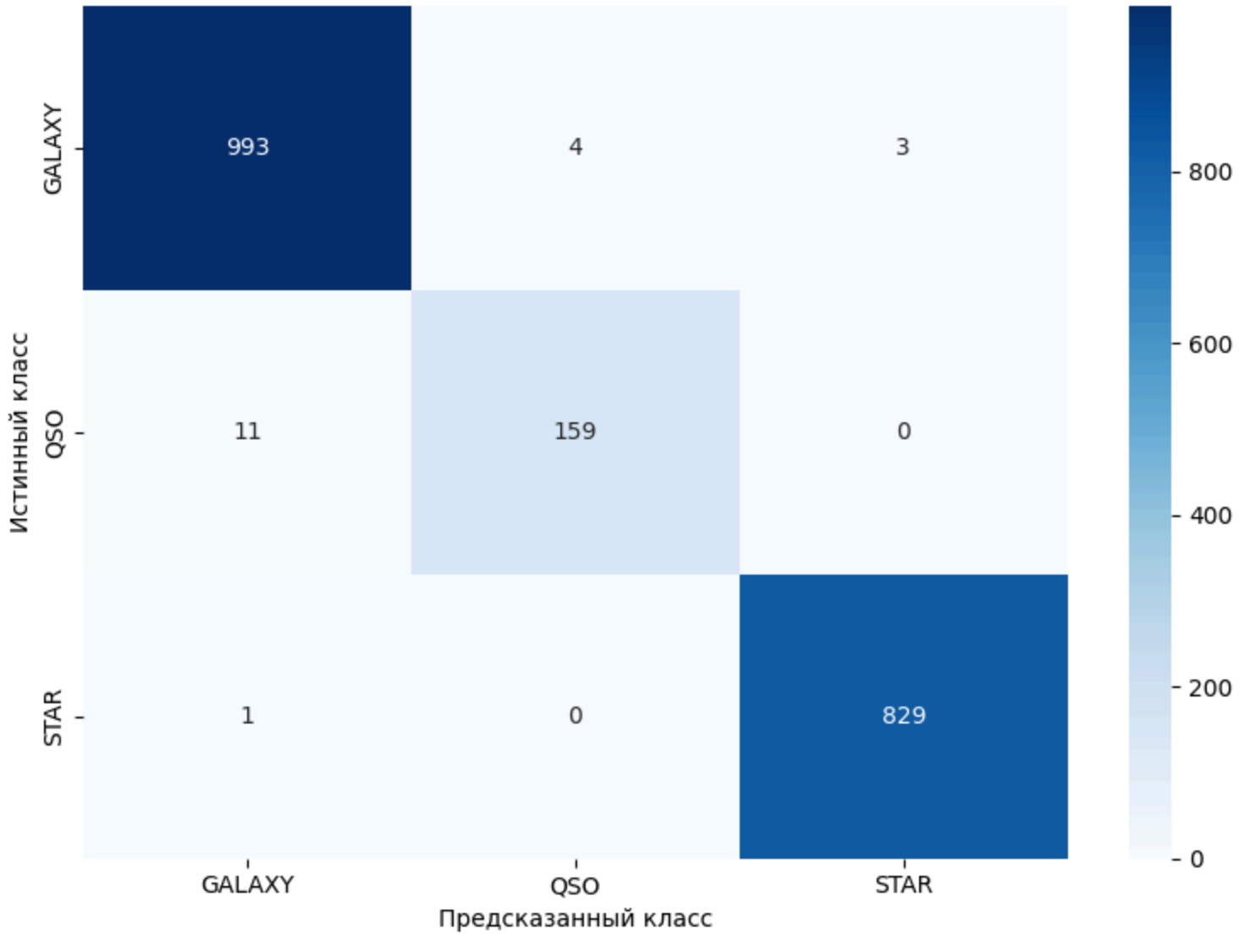
```
Результаты гипотезы 1:
1. Accuracy: 0.9900

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.986111 0.994000  0.990040
   QSO   0.981366 0.929412  0.954683
  STAR   0.996390 0.997590  0.996990

3. Macro F1: 0.9806

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     994    3     3
QSO         12  158     0
STAR         2    0   828
```



Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [12]:
```
from sklearn.model_selection import GridSearchCV

param_grid_h2 = {
    'max_depth': [5, 10, 15, 20, 25, 30, None],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10],
    'criterion': ['gini', 'entropy']
}

dt_for_tuning = DecisionTreeClassifier(random_state=42)
grid_search_h2 = GridSearchCV(
    dt_for_tuning,
    param_grid_h2,
    cv=5,
    scoring='f1_macro',
    n_jobs=-1
)
grid_search_h2.fit(X_train_class_h1_scaled, y_train_class_h1)

print("Лучшие параметры:")
for param, value in grid_search_h2.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nЛучший F1-score (кросс-валидация): {grid_search_h2.best_score_:.4f}")

dt_classifier_h2 = DecisionTreeClassifier(
    max_depth=grid_search_h2.best_params_['max_depth'],
    min_samples_split=grid_search_h2.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_h2.best_params_['min_samples_leaf'],
    criterion=grid_search_h2.best_params_['criterion'],
    random_state=42
)
dt_classifier_h2.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_h2 = dt_classifier_h2.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 2:")
class_metrics_h2 = evaluate_classification_model(y_test_class_h1, y_pred_class_h2, class_names)
```

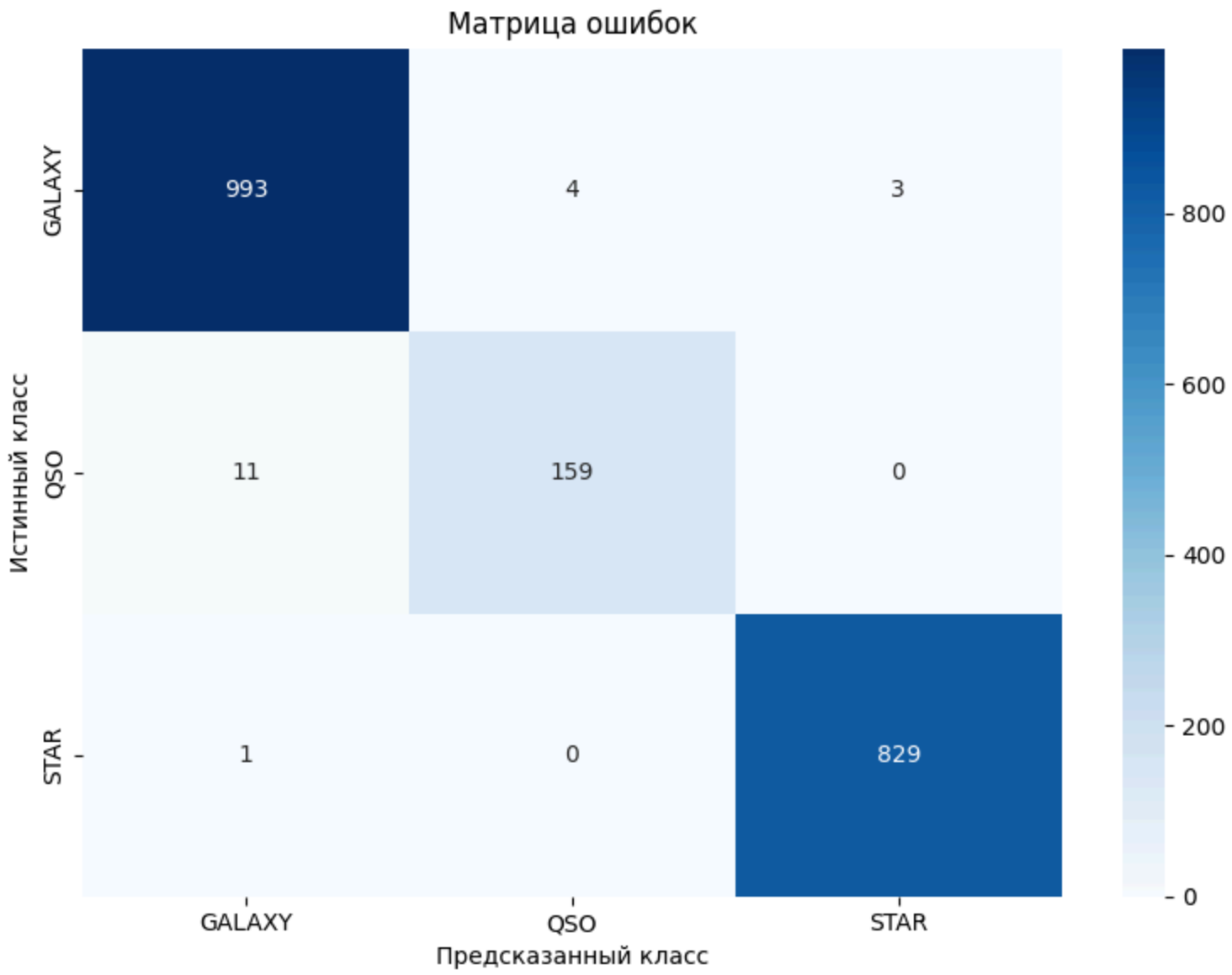
```
Лучшие параметры:
  criterion: gini
  max_depth: 5
  min_samples_leaf: 4
  min_samples_split: 2

Лучший F1-score (кросс-валидация): 0.9787

Результаты гипотезы 2:
1. Accuracy: 0.9905

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988060 0.993000  0.990524
   QSO   0.975460 0.935294  0.954955
  STAR   0.996394 0.998795  0.997593

3. Macro F1: 0.9810

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     993    4     3
QSO         11  159     0
STAR         1    0   829
```



Гипотеза 3: Добавление параметра max_features для ограничения количества признаков при разбиении улучшит качество модели.

In [13]:
```python
param_grid_h3 = {
    'max_depth': [grid_search_h2.best_params_['max_depth']],
    'min_samples_split': [grid_search_h2.best_params_['min_samples_split']],
    'min_samples_leaf': [grid_search_h2.best_params_['min_samples_leaf']],
    'criterion': [grid_search_h2.best_params_['criterion']],
    'max_features': ['sqrt', 'log2', None, 0.5, 0.7]
}

grid_search_h3 = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid_h3,
    cv=5,
    scoring='f1_macro',
    n_jobs=-1
)
grid_search_h3.fit(X_train_class_h1_scaled, y_train_class_h1)

print(f"Лучшие параметры: {grid_search_h3.best_params_}")
print(f"Лучший F1-score на кросс-валидации: {grid_search_h3.best_score_:.4f}")

dt_classifier_h3 = grid_search_h3.best_estimator_
y_pred_class_h3 = dt_classifier_h3.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 3:")
class_metrics_h3 = evaluate_classification_model(y_test_class_h1, y_pred_class_h3, class_names)
```

```
Лучшие параметры: {'criterion': 'gini', 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 4, 'min_s
amples_split': 2}
Лучший F1-score на кросс-валидации: 0.9787

Результаты гипотезы 3:
1. Accuracy: 0.9905

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988060 0.993000  0.990524
   QSO   0.975460 0.935294  0.954955
  STAR   0.996394 0.998795  0.997593

3. Macro F1: 0.9810

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     993    4     3
QSO         11  159     0
STAR         1    0   829
```



Сравним результаты всех гипотез с базовой моделью

In [14]:
```
comparison_hypotheses_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Гипотеза 1', 'Гипотеза 2', 'Гипотеза 3'],
    'Accuracy': [
        class_metrics['accuracy'],
        class_metrics_h1['accuracy'],
        class_metrics_h2['accuracy'],
        class_metrics_h3['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        class_metrics_h1['macro_f1'],
        class_metrics_h2['macro_f1'],
        class_metrics_h3['macro_f1']
    ]
})

print("Сравнение результатов гипотез для классификации:")
print(comparison_hypotheses_class.to_string(index=False))
```

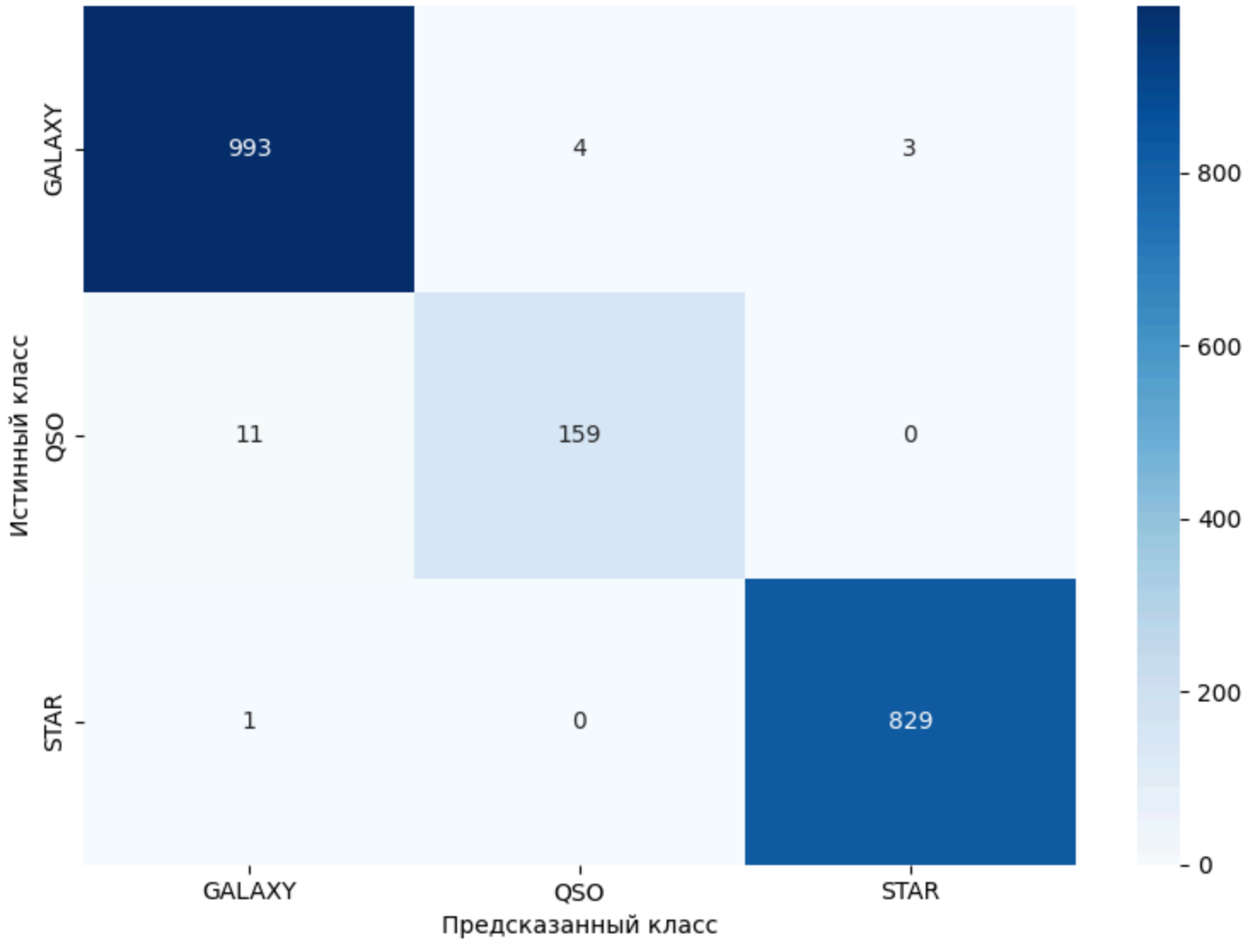
```
Сравнение результатов гипотез для классификации:
         Модель  Accuracy  Macro F1
Базовый бейзлайн    0.9885  0.978467
      Гипотеза 1    0.9900  0.980571
      Гипотеза 2    0.9905  0.981024
      Гипотеза 3    0.9905  0.981024
```

Сформируем улучшенный бейзлайн на основе лучшей гипотезы и обучим модель

In [15]:
```
dt_classifier_improved = DecisionTreeClassifier(
    max_depth=grid_search_h3.best_params_['max_depth'],
    min_samples_split=grid_search_h3.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_h3.best_params_['min_samples_leaf'],
```

```
    criterion=grid_search_h3.best_params_['criterion'],
    max_features=grid_search_h3.best_params_['max_features'],
    random_state=42
)
dt_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_improved = dt_classifier_improved.predict(X_test_class_h1_scaled)

print("Результаты улучшенного бейзлайна для классификации:")
class_metrics_improved = evaluate_classification_model(y_test_class_h1, y_pred_class_improved, class_names
```

```
Результаты улучшенного бейзлайна для классификации:
1. Accuracy: 0.9905

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988060 0.993000  0.990524
   QSO   0.975460 0.935294  0.954955
  STAR   0.996394 0.998795  0.997593

3. Macro F1: 0.9810

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     993    4     3
QSO         11  159     0
STAR         1    0   829
```



Сравним результаты улучшенного бейзлайна с базовым

In [16]:
```
comparison_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'Accuracy': [class_metrics['accuracy'], class_metrics_improved['accuracy']],
    'Macro F1': [class_metrics['macro_f1'], class_metrics_improved['macro_f1']]
})

print("Сравнение базового и улучшенного бейзлайна для классификации:")
print(comparison_class.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для классификации:
             Модель  Accuracy  Macro F1
   Базовый бейзлайн    0.9885  0.978467
Улучшенный бейзлайн    0.9905  0.981024
```

Регрессия

Гипотеза 1: Добавление стандартизации признаков улучшит качество модели.

In [17]:
```
scaler_reg_h1 = StandardScaler()
X_train_reg_h1_scaled = scaler_reg_h1.fit_transform(X_train_reg)
X_test_reg_h1_scaled = scaler_reg_h1.transform(X_test_reg)

dt_regressor_h1 = DecisionTreeRegressor(random_state=42)
dt_regressor_h1.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_h1 = dt_regressor_h1.predict(X_test_reg_h1_scaled)
```

```
print("Результаты гипотезы 1:")
reg_metrics_h1 = evaluate_regression_model(y_test_reg, y_pred_reg_h1)
```

```
Результаты гипотезы 1:
1. MAE: 2.1603

2. MSE: 9.6459

3. RMSE: 3.1058

4. R²: 0.1089
```

Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [18]:
```
param_grid_reg_h2 = {
    'max_depth': [5, 10, 15, 20, 25, 30, None],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10],
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error']
}

dt_reg_for_tuning = DecisionTreeRegressor(random_state=42)
grid_search_reg_h2 = GridSearchCV(
    dt_reg_for_tuning,
    param_grid_reg_h2,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search_reg_h2.fit(X_train_reg_h1_scaled, y_train_reg)

print("Лучшие параметры:")
for param, value in grid_search_reg_h2.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nЛучший MSE (кросс-валидация): {-grid_search_reg_h2.best_score_:.4f}")

dt_regressor_h2 = DecisionTreeRegressor(
    max_depth=grid_search_reg_h2.best_params_['max_depth'],
    min_samples_split=grid_search_reg_h2.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_reg_h2.best_params_['min_samples_leaf'],
    criterion=grid_search_reg_h2.best_params_['criterion'],
    random_state=42
)
dt_regressor_h2.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_h2 = dt_regressor_h2.predict(X_test_reg_h1_scaled)

print("\nРезультаты гипотезы 2:")
reg_metrics_h2 = evaluate_regression_model(y_test_reg, y_pred_reg_h2)
```

```
Лучшие параметры:
  criterion: squared_error
  max_depth: 5
  min_samples_leaf: 10
  min_samples_split: 2

Лучший MSE (кросс-валидация): 5.3595

Результаты гипотезы 2:
1. MAE: 1.6248

2. MSE: 5.3702

3. RMSE: 2.3174

4. R²: 0.5039
```

Гипотеза 3: Добавление параметра max_features для ограничения количества признаков при разбиении улучшит качество модели.

In [19]:
```
param_grid_reg_h3 = {
    'max_depth': [grid_search_reg_h2.best_params_['max_depth']],
    'min_samples_split': [grid_search_reg_h2.best_params_['min_samples_split']],
    'min_samples_leaf': [grid_search_reg_h2.best_params_['min_samples_leaf']],
    'criterion': [grid_search_reg_h2.best_params_['criterion']],
    'max_features': ['sqrt', 'log2', None, 0.5, 0.7]
}

grid_search_reg_h3 = GridSearchCV(
    DecisionTreeRegressor(random_state=42),
    param_grid_reg_h3,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search_reg_h3.fit(X_train_reg_h1_scaled, y_train_reg)

print(f"Лучшие параметры: {grid_search_reg_h3.best_params_}")
print(f"Лучший MSE на кросс-валидации: {-grid_search_reg_h3.best_score_:.4f}")
```

```
dt_regressor_h3 = grid_search_reg_h3.best_estimator_
y_pred_reg_h3 = dt_regressor_h3.predict(X_test_reg_h1_scaled)

print("\nРезультаты гипотезы 3:")
reg_metrics_h3 = evaluate_regression_model(y_test_reg, y_pred_reg_h3)
```

```
Лучшие параметры: {'criterion': 'squared_error', 'max_depth': 5, 'max_features': None, 'min_samples_leaf':
10, 'min_samples_split': 2}
Лучший MSE на кросс-валидации: 5.3595

Результаты гипотезы 3:
1. MAE: 1.6248

2. MSE: 5.3702

3. RMSE: 2.3174

4. R²: 0.5039
```

Сравним результаты всех гипотез с базовой моделью

In [20]:
```
comparison_hypotheses_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Гипотеза 1', 'Гипотеза 2', 'Гипотеза 3'],
    'MAE': [
        reg_metrics['mae'],
        reg_metrics_h1['mae'],
        reg_metrics_h2['mae'],
        reg_metrics_h3['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        reg_metrics_h1['rmse'],
        reg_metrics_h2['rmse'],
        reg_metrics_h3['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        reg_metrics_h1['r2'],
        reg_metrics_h2['r2'],
        reg_metrics_h3['r2']
    ]
})

print("Сравнение результатов гипотез для регрессии:")
print(comparison_hypotheses_reg.to_string(index=False))
```

```
Сравнение результатов гипотез для регрессии:
         Модель      MAE     RMSE       R²
Базовый бейзлайн 2.178230 3.133589 0.092916
      Гипотеза 1 2.160287 3.105790 0.108938
      Гипотеза 2 1.624829 2.317374 0.503915
      Гипотеза 3 1.624829 2.317374 0.503915
```

Сформируем улучшенный бейзлайн на основе лучшей гипотезы и обучим модель

In [21]:
```
dt_regressor_improved = DecisionTreeRegressor(
    max_depth=grid_search_reg_h3.best_params_['max_depth'],
    min_samples_split=grid_search_reg_h3.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_reg_h3.best_params_['min_samples_leaf'],
    criterion=grid_search_reg_h3.best_params_['criterion'],
    max_features=grid_search_reg_h3.best_params_['max_features'],
    random_state=42
)
dt_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_improved = dt_regressor_improved.predict(X_test_reg_h1_scaled)

print("Результаты улучшенного бейзлайна для регрессии:")
reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_reg_improved)
```

```
Результаты улучшенного бейзлайна для регрессии:
1. MAE: 1.6248

2. MSE: 5.3702

3. RMSE: 2.3174

4. R²: 0.5039
```

Сравним результаты улучшенного бейзлайна с базовым

In [22]:
```
comparison_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'MAE': [reg_metrics['mae'], reg_metrics_improved['mae']],
    'RMSE': [reg_metrics['rmse'], reg_metrics_improved['rmse']],
    'R²': [reg_metrics['r2'], reg_metrics_improved['r2']]
})

print("Сравнение базового и улучшенного бейзлайна для регрессии:")
print(comparison_reg.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для регрессии:
            Модель      MAE     RMSE       R²
   Базовый бейзлайн 2.178230 3.133589 0.092916
Улучшенный бейзлайн 1.624829 2.317374 0.503915
```

4. Имплементация алгоритма машинного обучения

Перейдем к имлпементации алгоритмов

Классификация

Реализуем алгоритм DecisionTree для классификации

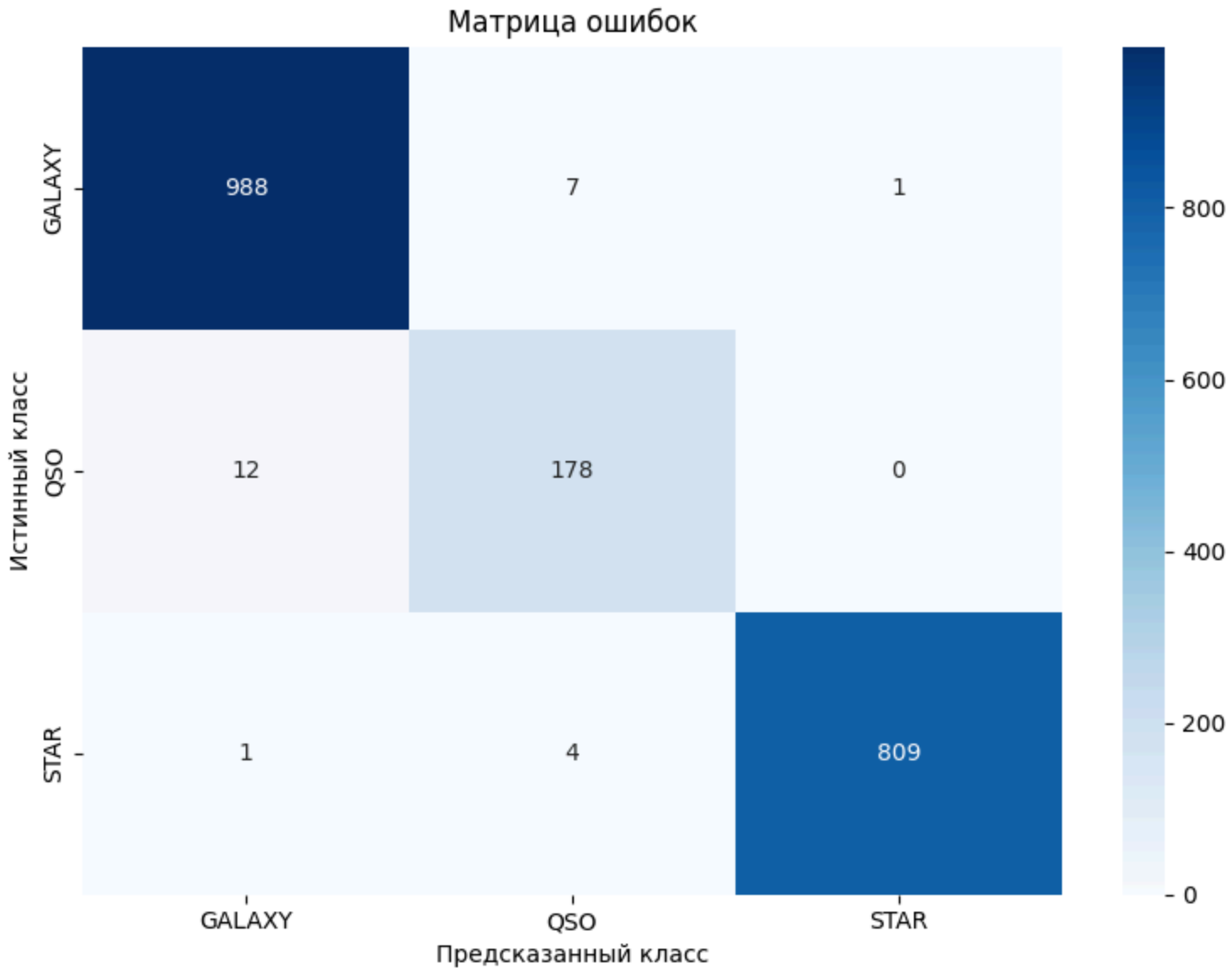
In [23]:
```python
class MyDecisionTreeClassifier:
    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf=1,
                 criterion='gini', max_features=None, random_state=None):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.max_features = max_features
        self.random_state = random_state
        self.tree = None
        self.n_features_ = None
        self.classes_ = None

    def _gini(self, y):
        if len(y) == 0:
            return 0
        counts = np.bincount(y)
        probabilities = counts / len(y)
        return 1 - np.sum(probabilities ** 2)

    def _entropy(self, y):
        if len(y) == 0:
            return 0
        counts = np.bincount(y)
        probabilities = counts[counts > 0] / len(y)
        return -np.sum(probabilities * np.log2(probabilities))

    def _impurity(self, y):
        if self.criterion == 'gini':
            return self._gini(y)
        elif self.criterion == 'entropy':
            return self._entropy(y)
        else:
            raise ValueError(f"Unknown criterion: {self.criterion}")

    def _best_split(self, X, y):
        best_gain = -1
        best_feature = None
        best_threshold = None

        n_features = X.shape[1]
        if self.max_features is not None:
            if isinstance(self.max_features, (int, float)):
                if isinstance(self.max_features, float):
                    n_features = max(1, int(self.max_features * n_features))
                else:
                    n_features = min(self.max_features, n_features)
            elif self.max_features == 'sqrt':
                n_features = int(np.sqrt(n_features))
            elif self.max_features == 'log2':
                n_features = int(np.log2(n_features))

        if self.random_state is not None:
            np.random.seed(self.random_state)
        features = np.random.choice(X.shape[1], n_features, replace=False)

        parent_impurity = self._impurity(y)

        for feature in features:
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = ~left_indices

                if np.sum(left_indices) < self.min_samples_leaf or np.sum(right_indices) < self.min_samples
                    continue

                left_impurity = self._impurity(y[left_indices])
                right_impurity = self._impurity(y[right_indices])

                n_left = np.sum(left_indices)
                n_right = np.sum(right_indices)
                n_total = len(y)
```

```
                weighted_impurity = (n_left / n_total) * left_impurity + (n_right / n_total) * right_impuri
                gain = parent_impurity - weighted_impurity

                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold, best_gain

    def _build_tree(self, X, y, depth=0):
        n_samples = len(y)
        n_classes = len(np.unique(y))

        if (self.max_depth is not None and depth >= self.max_depth) or \
           n_samples < self.min_samples_split or \
           n_classes == 1:
            return {'class': np.bincount(y).argmax(), 'is_leaf': True}

        feature, threshold, gain = self._best_split(X, y)

        if feature is None or gain <= 0:
            return {'class': np.bincount(y).argmax(), 'is_leaf': True}

        left_indices = X[:, feature] <= threshold
        right_indices = ~left_indices

        if np.sum(left_indices) < self.min_samples_leaf or np.sum(right_indices) < self.min_samples_leaf:
            return {'class': np.bincount(y).argmax(), 'is_leaf': True}

        left_tree = self._build_tree(X[left_indices], y[left_indices], depth + 1)
        right_tree = self._build_tree(X[right_indices], y[right_indices], depth + 1)

        return {
            'feature': feature,
            'threshold': threshold,
            'left': left_tree,
            'right': right_tree,
            'is_leaf': False
        }

    def _predict_sample(self, x, node):
        if node['is_leaf']:
            return node['class']

        if x[node['feature']] <= node['threshold']:
            return self._predict_sample(x, node['left'])
        else:
            return self._predict_sample(x, node['right'])

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)

        self.n_features_ = X.shape[1]
        self.classes_ = np.unique(y)

        self.tree = self._build_tree(X, y)
        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []
        for x in X:
            predictions.append(self._predict_sample(x, self.tree))
        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [24]:
```
my_dt_classifier = MyDecisionTreeClassifier(random_state=42)
my_dt_classifier.fit(X_train_class.values, y_train_class)
y_pred_my_class = my_dt_classifier.predict(X_test_class.values)

print("Результаты имплементированной модели классификации:")
my_class_metrics = evaluate_classification_model(y_test_class, y_pred_my_class, class_names)
```

```
Результаты имплементированной модели классификации:
1. Accuracy: 0.9875

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.987013 0.991968  0.989484
   QSO   0.941799 0.936842  0.939314
  STAR   0.998765 0.993857  0.996305

3. Macro F1: 0.9750

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     988    7     1
QSO         12  178     0
STAR         1    4   809
```



Сравним результаты имплементированной модели с базовым бейзлайном

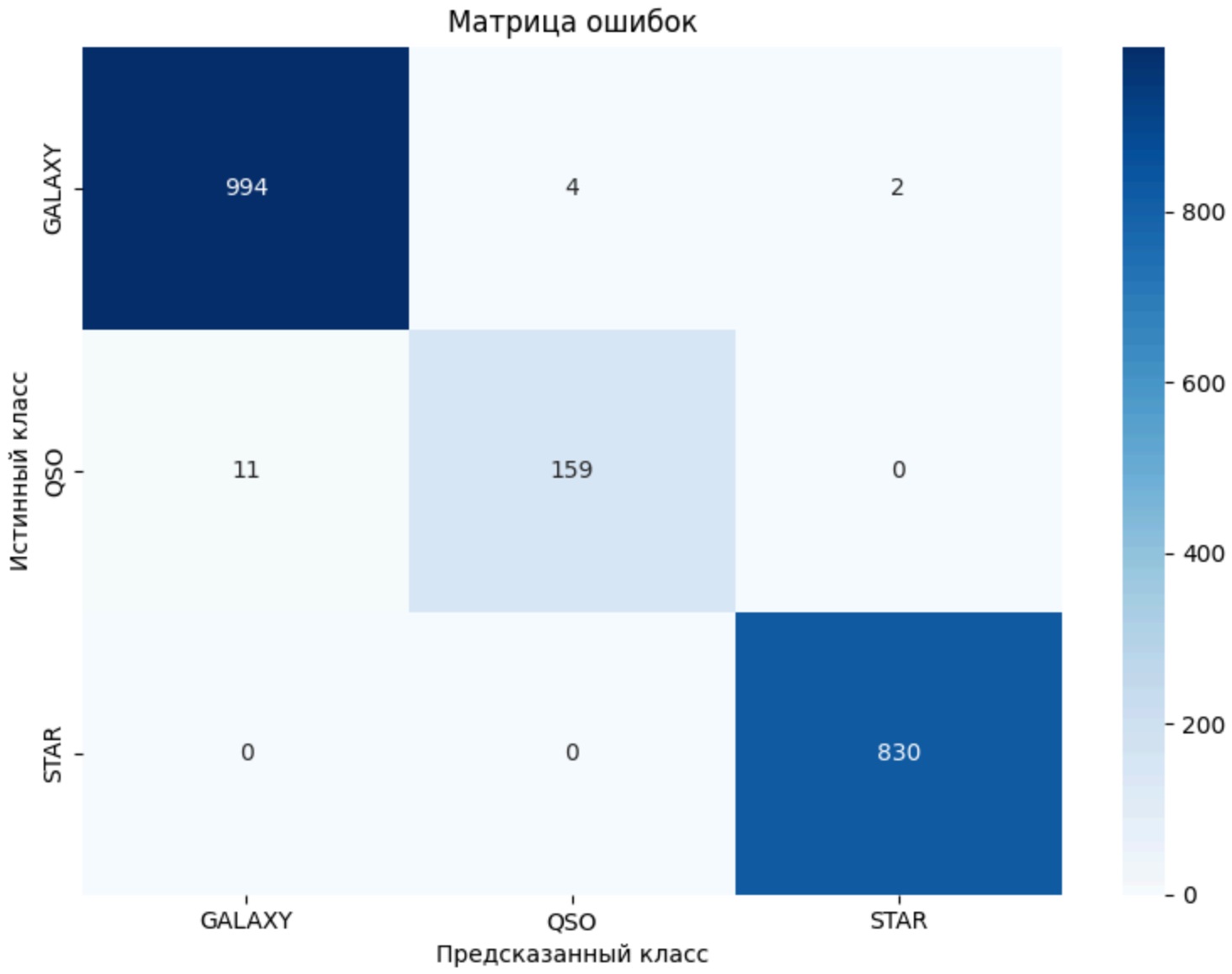
In [25]:
```
comparison_my_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель'],
    'Accuracy': [class_metrics['accuracy'], my_class_metrics['accuracy']],
    'Macro F1': [class_metrics['macro_f1'], my_class_metrics['macro_f1']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_class.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                  Модель  Accuracy  Macro F1
Базовый бейзлайн (sklearn)    0.9885  0.978467
 Имплементированная модель    0.9875  0.975035
```

Теперь применим техники из улучшенного бейзлайна

In [26]:
```
my_dt_classifier_improved = MyDecisionTreeClassifier(
    max_depth=grid_search_h3.best_params_['max_depth'],
    min_samples_split=grid_search_h3.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_h3.best_params_['min_samples_leaf'],
    criterion=grid_search_h3.best_params_['criterion'],
    max_features=grid_search_h3.best_params_['max_features'],
    random_state=42
)
my_dt_classifier_improved.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_my_class_improved = my_dt_classifier_improved.predict(X_test_class_h1_scaled)

print("Результаты имплементированной модели с улучшениями:")
my_class_metrics_improved = evaluate_classification_model(y_test_class_h1, y_pred_my_class_improved, class_
```

```
Результаты имплементированной модели с улучшениями:
1. Accuracy: 0.9915

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.989055 0.994000  0.991521
   QSO   0.975460 0.935294  0.954955
  STAR   0.997596 1.000000  0.998797

3. Macro F1: 0.9818

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     994    4     2
QSO         11  159     0
STAR         0    0   830
```



Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [27]:
```python
comparison_my_class_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'Accuracy': [class_metrics_improved['accuracy'], my_class_metrics_improved['accuracy']],
    'Macro F1': [class_metrics_improved['macro_f1'], my_class_metrics_improved['macro_f1']]
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:")
print(comparison_my_class_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                              Модель  Accuracy  Macro F1
          Улучшенный бейзлайн (sklearn)    0.9905  0.981024
Имплементированная модель с улучшениями    0.9915  0.981758
```

Регрессия

Реализуем алгоритм DecisionTree для регрессии

In [28]:
```python
class MyDecisionTreeRegressor:
    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf=1,
                 criterion='squared_error', max_features=None, random_state=None):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.max_features = max_features
        self.random_state = random_state
        self.tree = None
        self.n_features_ = None

    def _mse(self, y):
        if len(y) == 0:
            return 0
        mean = np.mean(y)
        return np.mean((y - mean) ** 2)
```

```
    def _mae(self, y):
        if len(y) == 0:
            return 0
        median = np.median(y)
        return np.mean(np.abs(y - median))

    def _impurity(self, y):
        if self.criterion == 'squared_error' or self.criterion == 'friedman_mse':
            return self._mse(y)
        elif self.criterion == 'absolute_error':
            return self._mae(y)
        else:
            raise ValueError(f"Unknown criterion: {self.criterion}")

    def _best_split(self, X, y):
        best_gain = -1
        best_feature = None
        best_threshold = None

        n_features = X.shape[1]
        if self.max_features is not None:
            if isinstance(self.max_features, (int, float)):
                if isinstance(self.max_features, float):
                    n_features = max(1, int(self.max_features * n_features))
                else:
                    n_features = min(self.max_features, n_features)
            elif self.max_features == 'sqrt':
                n_features = int(np.sqrt(n_features))
            elif self.max_features == 'log2':
                n_features = int(np.log2(n_features))

        if self.random_state is not None:
            np.random.seed(self.random_state)
        features = np.random.choice(X.shape[1], n_features, replace=False)

        parent_impurity = self._impurity(y)

        for feature in features:
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = ~left_indices

                if np.sum(left_indices) < self.min_samples_leaf or np.sum(right_indices) < self.min_samples
                    continue

                left_impurity = self._impurity(y[left_indices])
                right_impurity = self._impurity(y[right_indices])

                n_left = np.sum(left_indices)
                n_right = np.sum(right_indices)
                n_total = len(y)

                weighted_impurity = (n_left / n_total) * left_impurity + (n_right / n_total) * right_impur
                gain = parent_impurity - weighted_impurity

                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold, best_gain

    def _build_tree(self, X, y, depth=0):
        n_samples = len(y)

        if (self.max_depth is not None and depth >= self.max_depth) or \
           n_samples < self.min_samples_split:
            return {'value': np.mean(y), 'is_leaf': True}

        feature, threshold, gain = self._best_split(X, y)

        if feature is None or gain <= 0:
            return {'value': np.mean(y), 'is_leaf': True}

        left_indices = X[:, feature] <= threshold
        right_indices = ~left_indices

        if np.sum(left_indices) < self.min_samples_leaf or np.sum(right_indices) < self.min_samples_leaf:
            return {'value': np.mean(y), 'is_leaf': True}

        left_tree = self._build_tree(X[left_indices], y[left_indices], depth + 1)
        right_tree = self._build_tree(X[right_indices], y[right_indices], depth + 1)

        return {
            'feature': feature,
            'threshold': threshold,
            'left': left_tree,
            'right': right_tree,
```

```
            'is_leaf': False
        }

    def _predict_sample(self, x, node):
        if node['is_leaf']:
            return node['value']

        if x[node['feature']] <= node['threshold']:
            return self._predict_sample(x, node['left'])
        else:
            return self._predict_sample(x, node['right'])

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)

        self.n_features_ = X.shape[1]

        self.tree = self._build_tree(X, y)
        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []
        for x in X:
            predictions.append(self._predict_sample(x, self.tree))
        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [29]:
```
my_dt_regressor = MyDecisionTreeRegressor(random_state=42)
my_dt_regressor.fit(X_train_reg.values, y_train_reg.values)
y_pred_my_reg = my_dt_regressor.predict(X_test_reg.values)

print("Результаты имплементированной модели регрессии:")
my_reg_metrics = evaluate_regression_model(y_test_reg, y_pred_my_reg)
```

```
Результаты имплементированной модели регрессии:
1. MAE: 2.2165

2. MSE: 9.9677

3. RMSE: 3.1572

4. R²: 0.0792
```

Сравним результаты имплементированной модели с базовым бейзлайном

In [30]:
```
comparison_my_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель'],
    'MAE': [reg_metrics['mae'], my_reg_metrics['mae']],
    'RMSE': [reg_metrics['rmse'], my_reg_metrics['rmse']],
    'R²': [reg_metrics['r2'], my_reg_metrics['r2']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_reg.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                   Модель      MAE     RMSE       R²
Базовый бейзлайн (sklearn) 2.178230 3.133589 0.092916
Имплементированная модель 2.216507 3.157167 0.079214
```

Теперь применим техники из улучшенного бейзлайна

In [31]:
```
my_dt_regressor_improved = MyDecisionTreeRegressor(
    max_depth=grid_search_reg_h3.best_params_['max_depth'],
    min_samples_split=grid_search_reg_h3.best_params_['min_samples_split'],
    min_samples_leaf=grid_search_reg_h3.best_params_['min_samples_leaf'],
    criterion=grid_search_reg_h3.best_params_['criterion'],
    max_features=grid_search_reg_h3.best_params_['max_features'],
    random_state=42
)
my_dt_regressor_improved.fit(X_train_reg_h1_scaled, y_train_reg.values)
y_pred_my_reg_improved = my_dt_regressor_improved.predict(X_test_reg_h1_scaled)

print("Результаты имплементированной модели с улучшениями:")
my_reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_my_reg_improved)
```

```
Результаты имплементированной модели с улучшениями:
1. MAE: 1.6260

2. MSE: 5.3718

3. RMSE: 2.3177

4. R²: 0.5038
```

Сравним результаты имплементированной модели с улучшениями с улучшенным бейзлайном

In [32]:
```
comparison_my_reg_improved = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'MAE': [reg_metrics_improved['mae'], my_reg_metrics_improved['mae']],
    'RMSE': [reg_metrics_improved['rmse'], my_reg_metrics_improved['rmse']],
    'R²': [reg_metrics_improved['r2'], my_reg_metrics_improved['r2']]
})

print("Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:")
print(comparison_my_reg_improved.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшениями с улучшенным бейзлайном:
                              Модель      MAE     RMSE       R²
         Улучшенный бейзлайн (sklearn) 1.624829 2.317374 0.503915
Имплементированная модель с улучшениями 1.625966 2.317720 0.503767
```

Общие выводы по результатам всех моделей

Сравним все 4 модели для классификации и регрессии: базовый бейзлайн из sklearn, имплементированную модель базового бейзлайна, модель с улучшенным бейзлайном из sklearn и имплементированную модель улучшенного бейзлайна.

Классификация

In [33]:
```
final_comparison_class = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'Accuracy': [
        class_metrics['accuracy'],
        my_class_metrics['accuracy'],
        class_metrics_improved['accuracy'],
        my_class_metrics_improved['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        my_class_metrics['macro_f1'],
        class_metrics_improved['macro_f1'],
        my_class_metrics_improved['macro_f1']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ")
print(final_comparison_class.to_string(index=False))

print("\nВЫВОДЫ ПО КЛАССИФИКАЦИИ:")
print(f"1. Базовый бейзлайн (sklearn): Accuracy = {class_metrics['accuracy']:.4f}, Macro F1 = {class_metri
print(f"2. Имплементированная модель базового бейзлайна: Accuracy = {my_class_metrics['accuracy']:.4f}, Ma
print(f"3. Улучшенный бейзлайн (sklearn): Accuracy = {class_metrics_improved['accuracy']:.4f}, Macro F1 = 
print(f"4. Имплементированная модель улучшенного бейзлайна: Accuracy = {my_class_metrics_improved['accuracy
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  Accuracy: {((class_metrics_improved['accuracy'] - class_metrics['accuracy']) / class_metrics['ac
print(f"  Macro F1: {((class_metrics_improved['macro_f1'] - class_metrics['macro_f1']) / class_metrics['ma
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в Accuracy: {abs(class_metrics_improved['accuracy'] - my_class_metrics_improved['accuracy
print(f"  Разница в Macro F1: {abs(class_metrics_improved['macro_f1'] - my_class_metrics_improved['macro_f1
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ
                                        Модель  Accuracy  Macro F1
                          Базовый бейзлайн (sklearn)    0.9885  0.978467
       Имплементированная модель базового бейзлайна    0.9875  0.975035
                       Улучшенный бейзлайн (sklearn)    0.9905  0.981024
Имплементированная модель улучшенного бейзлайна    0.9915  0.981758

ВЫВОДЫ ПО КЛАССИФИКАЦИИ:
1. Базовый бейзлайн (sklearn): Accuracy = 0.9885, Macro F1 = 0.9785
2. Имплементированная модель базового бейзлайна: Accuracy = 0.9875, Macro F1 = 0.9750
3. Улучшенный бейзлайн (sklearn): Accuracy = 0.9905, Macro F1 = 0.9810
4. Имплементированная модель улучшенного бейзлайна: Accuracy = 0.9915, Macro F1 = 0.9818

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  Accuracy: 0.20%
  Macro F1: 0.26%

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в Accuracy: 0.001000
  Разница в Macro F1: 0.000734
```

Регрессия

In [34]:
```
final_comparison_reg = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
```

```
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'MAE': [
        reg_metrics['mae'],
        my_reg_metrics['mae'],
        reg_metrics_improved['mae'],
        my_reg_metrics_improved['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        my_reg_metrics['rmse'],
        reg_metrics_improved['rmse'],
        my_reg_metrics_improved['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        my_reg_metrics['r2'],
        reg_metrics_improved['r2'],
        my_reg_metrics_improved['r2']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ")
print(final_comparison_reg.to_string(index=False))

print("\nВЫВОДЫ ПО РЕГРЕССИИ:")
print(f"1. Базовый бейзлайн (sklearn): MAE = {reg_metrics['mae']:.4f}, RMSE = {reg_metrics['rmse']:.4f}, R
print(f"2. Имплементированная модель базового бейзлайна: MAE = {my_reg_metrics['mae']:.4f}, RMSE = {my_reg_
print(f"3. Улучшенный бейзлайн (sklearn): MAE = {reg_metrics_improved['mae']:.4f}, RMSE = {reg_metrics_imp
print(f"4. Имплементированная модель улучшенного бейзлайна: MAE = {my_reg_metrics_improved['mae']:.4f}, RMS
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  MAE: {((reg_metrics['mae'] - reg_metrics_improved['mae']) / reg_metrics['mae'] * 100):.2f}% улучш
print(f"  RMSE: {((reg_metrics['rmse'] - reg_metrics_improved['rmse']) / reg_metrics['rmse'] * 100):.2f}% у
print(f"  R²: {((reg_metrics_improved['r2'] - reg_metrics['r2']) / abs(reg_metrics['r2']) * 100):.2f}% улуч
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в MAE: {abs(reg_metrics_improved['mae'] - my_reg_metrics_improved['mae']):.6f}")
print(f"  Разница в RMSE: {abs(reg_metrics_improved['rmse'] - my_reg_metrics_improved['rmse']):.6f}")
print(f"  Разница в R²: {abs(reg_metrics_improved['r2'] - my_reg_metrics_improved['r2']):.6f}")
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ
                                      Модель      MAE     RMSE       R²
                     Базовый бейзлайн (sklearn) 2.178230 3.133589 0.092916
   Имплементированная модель базового бейзлайна 2.216507 3.157167 0.079214
                  Улучшенный бейзлайн (sklearn) 1.624829 2.317374 0.503915
Имплементированная модель улучшенного бейзлайна 1.625966 2.317720 0.503767

ВЫВОДЫ ПО РЕГРЕССИИ:
1. Базовый бейзлайн (sklearn): MAE = 2.1782, RMSE = 3.1336, R² = 0.0929
2. Имплементированная модель базового бейзлайна: MAE = 2.2165, RMSE = 3.1572, R² = 0.0792
3. Улучшенный бейзлайн (sklearn): MAE = 1.6248, RMSE = 2.3174, R² = 0.5039
4. Имплементированная модель улучшенного бейзлайна: MAE = 1.6260, RMSE = 2.3177, R² = 0.5038

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  MAE: 25.41% улучшение
  RMSE: 26.05% улучшение
  R²: 442.34% улучшение

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в MAE: 0.001137
  Разница в RMSE: 0.000346
  Разница в R²: 0.000148
```

Лабораторная работа №4 (Проведение исследований со случайным лесом)

2. Создание бейзлайна и оценка качества

Перейдем к созданию базовых моделей

Классификация

Загрузим датасет и посмотрим на данные

In [1]:
```
import pandas as pd

df_class = pd.read_csv('data/Skyserver_SQL2_27_2018 6_51_39 PM.csv')

print(f"Размерность данных")
print(df_class.shape)
print(f"\nИнформация о данных")
print(df_class.info())
print(f"\nПервые 5 строк")
print(df_class.head())
print(f"\nРаспределение классов")
print(df_class['class'].value_counts())
```

```
Размерность данных
(10000, 18)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   objid      10000 non-null  float64
 1   ra         10000 non-null  float64
 2   dec        10000 non-null  float64
 3   u          10000 non-null  float64
 4   g          10000 non-null  float64
 5   r          10000 non-null  float64
 6   i          10000 non-null  float64
 7   z          10000 non-null  float64
 8   run        10000 non-null  int64
 9   rerun      10000 non-null  int64
 10  camcol     10000 non-null  int64
 11  field      10000 non-null  int64
 12  specobjid  10000 non-null  float64
 13  class      10000 non-null  object
 14  redshift   10000 non-null  float64
 15  plate      10000 non-null  int64
 16  mjd        10000 non-null  int64
 17  fiberid    10000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 1.4+ MB
None

Первые 5 строк
          objid          ra       dec         u         g         r
i  \
0  1.237650e+18  183.531326  0.089693  19.47406  17.04240  15.94699  15.50
342
1  1.237650e+18  183.598370  0.135285  18.66280  17.21449  16.67637  16.48
922
2  1.237650e+18  183.680207  0.126185  19.38298  18.19169  17.47428  17.08
732
3  1.237650e+18  183.870529  0.049911  17.76536  16.60272  16.16116  15.98
233
4  1.237650e+18  183.883288  0.102557  17.55025  16.26342  16.43869  16.55
492

          z  run  rerun  camcol  field     specobjid   class  redshift  pl
ate  \
0  15.22531  752    301       4    267  3.722360e+18    STAR -0.000009   3
306
1  16.39150  752    301       4    267  3.638140e+17    STAR -0.000055
323
2  16.80125  752    301       4    268  3.232740e+17  GALAXY  0.123111
287
3  15.90438  752    301       4    269  3.722370e+18    STAR -0.000111   3
306
4  16.61326  752    301       4    269  3.722370e+18    STAR  0.000590   3
306

     mjd  fiberid
0  54922      491
```

```
1  51615      541
2  52023      513
3  54922      510
4  54922      512

Распределение классов
class
GALAXY    4998
STAR      4152
QSO        850
Name: count, dtype: int64
```

Выделим исходные признаки, которые непосредственно описывают физические свойства объектов и целевую переменную. А также закодируем таргет, так как это категориальный признак.

In [2]:
```
from sklearn.preprocessing import LabelEncoder

X_class = df_class[['u', 'g', 'r', 'i', 'z', 'redshift']]
y_class = df_class['class']

le = LabelEncoder()
y_class_encoded = le.fit_transform(y_class)
class_names = le.classes_
```

Разделим данные на выборку для обучения и тестовую выборку.

In [3]:
```
from sklearn.model_selection import train_test_split

X_train_class, X_test_class, y_train_class, y_test_class = train_test_spl
    X_class, y_class_encoded, test_size=0.2, random_state=42
)
```

Обучим базовую модель Random Forest и выполним предсказания на тестовой выборке.

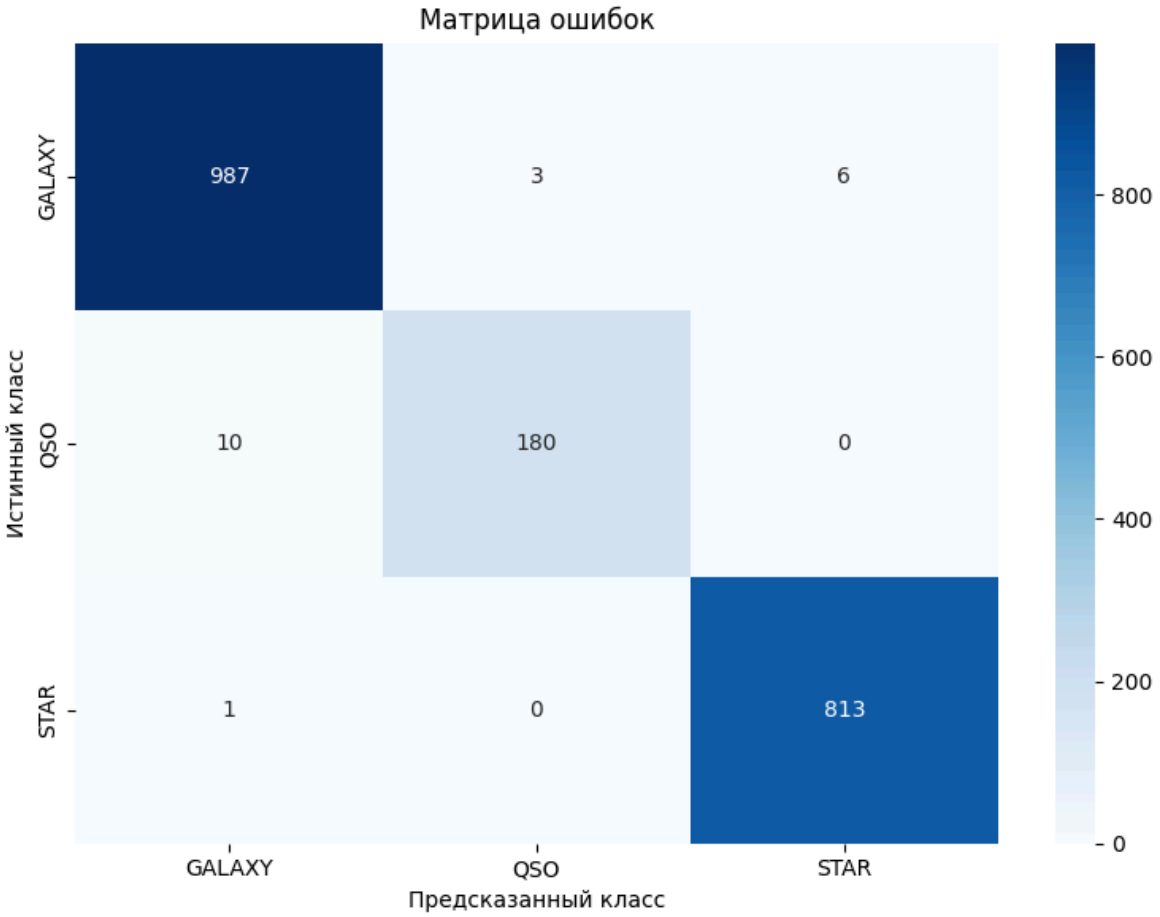
In [4]:
```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_class, y_train_class)

y_pred_class = rf_classifier.predict(X_test_class)
```

Опишем функцию, которая будет использоваться для оценки обученной модели классификации.

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix
)

def evaluate_classification_model(y_true, y_pred, class_names):
```

```
    accuracy = accuracy_score(y_true, y_pred)
    print(f"1. Accuracy: {accuracy:.4f}")

    print(f"\n2. Метрики по классам:")
    precision = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)

    metrics_df = pd.DataFrame({
        'Класс': class_names,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
    print(metrics_df.to_string(index=False))

    macro_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"\n3. Macro F1: {macro_f1:.4f}")

    print(f"\n4. Матрица ошибок:")
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
    print(cm_df)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Матрица ошибок')
    plt.ylabel('Истинный класс')
    plt.xlabel('Предсказанный класс')
    plt.tight_layout()
    plt.show()

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'macro_f1': macro_f1,
        'confusion_matrix': cm
    }

class_metrics = evaluate_classification_model(y_test_class, y_pred_class,
```

```
1. Accuracy: 0.9900

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988978 0.990964  0.989970
   QSO   0.983607 0.947368  0.965147
  STAR   0.992674 0.998771  0.995713

3. Macro F1: 0.9836

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     987    3     6
QSO         10  180     0
STAR         1    0   813
```



Регрессия

Загрузим датасет и посмотрим на данные

In [6]:
```
df_reg = pd.read_csv('data/abalone.csv')

print(f"Размерность данных")
print(df_reg.shape)
print(f"\nИнформация о данных")
print(df_reg.info())
print(f"\nПервые 5 строк")
print(df_reg.head())
print(f"\nСтатистика по числовым признакам")
print(df_reg.describe())
```

```
Размерность данных
(4177, 9)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None

Первые 5 строк
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera wei
ght  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1
010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0
485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1
415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1
140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0
395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7

Статистика по числовым признакам
            Length     Diameter       Height  Whole weight  Shucked weight
\
count  4177.000000  4177.000000  4177.000000   4177.000000     4177.000000
mean      0.523992     0.407881     0.139516      0.828742        0.359367
std       0.120093     0.099240     0.041827      0.490389        0.221963
min       0.075000     0.055000     0.000000      0.002000        0.001000
25%       0.450000     0.350000     0.115000      0.441500        0.186000
50%       0.545000     0.425000     0.140000      0.799500        0.336000
75%       0.615000     0.480000     0.165000      1.153000        0.502000
max       0.815000     0.650000     1.130000      2.825500        1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
```

```
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Выделим признаки и целевую переменную. Закодируем категориальный признак Sex.

In [7]:
```
X_reg = df_reg.drop('Rings', axis=1)
y_reg = df_reg['Rings']

le_sex = LabelEncoder()
X_reg_encoded = X_reg.copy()
X_reg_encoded['Sex'] = le_sex.fit_transform(X_reg['Sex'])
```

Разделим данные на выборку для обучения и тестовую выборку.

In [8]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg_encoded, y_reg, test_size=0.2, random_state=42
)
```

Обучим базовую модель Random Forest и выполним предсказания на тестовой выборке.

In [9]:
```
from sklearn.ensemble import RandomForestRegressor

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = rf_regressor.predict(X_test_reg)
```

Опишем функцию, которая будет использоваться для оценки обученной модели регрессии.

In [10]:
```
import numpy as np
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
)

def evaluate_regression_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    print(f"1. MAE: {mae:.4f}")

    mse = mean_squared_error(y_true, y_pred)
    print(f"\n2. MSE: {mse:.4f}")

    rmse = np.sqrt(mse)
    print(f"\n3. RMSE: {rmse:.4f}")

    r2 = r2_score(y_true, y_pred)
    print(f"\n4. R²: {r2:.4f}")

    return {
        'mae': mae,
        'mse': mse,
```

```
        'rmse': rmse,
        'r2': r2
    }

reg_metrics = evaluate_regression_model(y_test_reg, y_pred_reg)
```

```
1. MAE: 1.5899

2. MSE: 5.0915

3. RMSE: 2.2564

4. R²: 0.5297
```

3. Улучшение бейзлайна

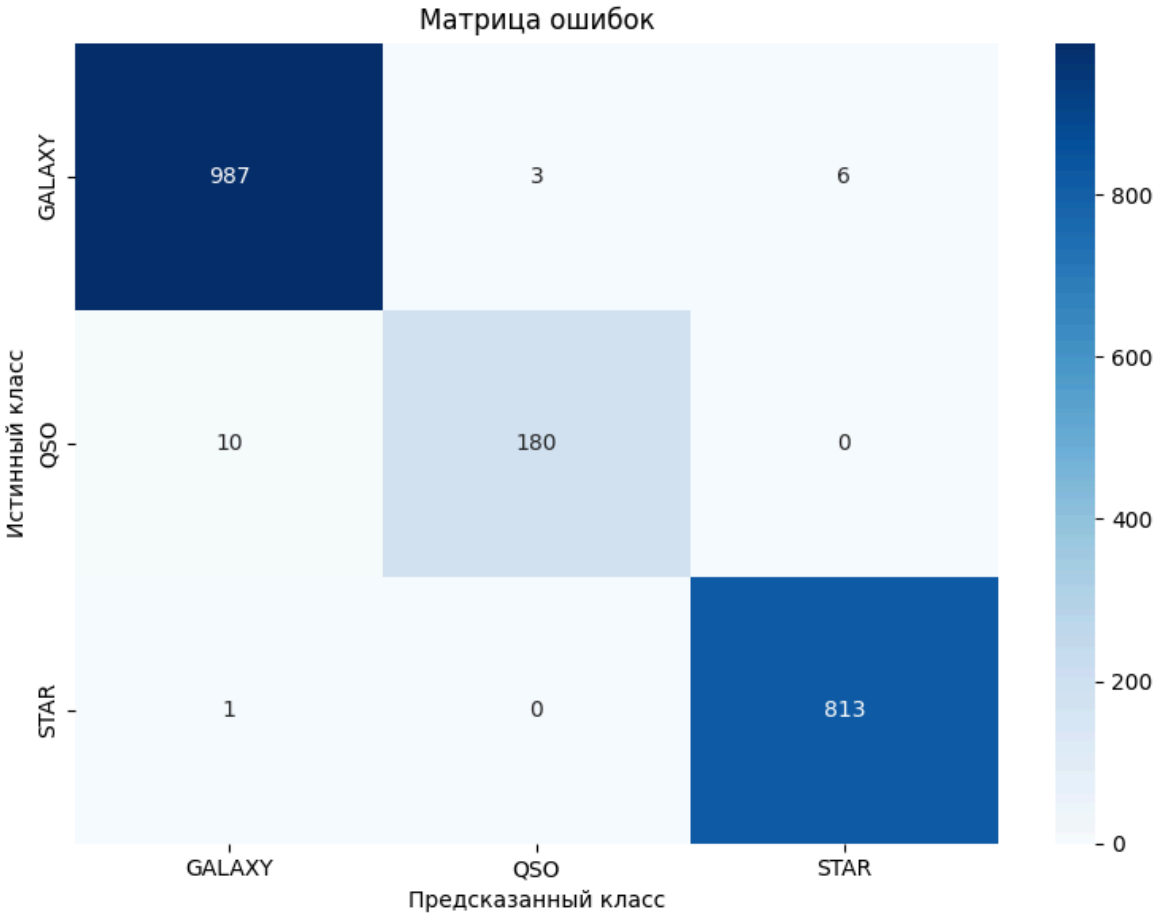
Перейдем к формулированию и проверкам гипотез

Классификация

Гипотеза 1: Стандартизация признаков улучшит качество модели.

In [11]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, StratifiedKFold

scaler = StandardScaler()
X_train_class_scaled = scaler.fit_transform(X_train_class)
X_test_class_scaled = scaler.transform(X_test_class)

rf_classifier_scaled = RandomForestClassifier(n_estimators=100, random_st
rf_classifier_scaled.fit(X_train_class_scaled, y_train_class)
y_pred_class_scaled = rf_classifier_scaled.predict(X_test_class_scaled)

print("Результаты с стандартизацией:")
metrics_scaled = evaluate_classification_model(y_test_class, y_pred_class
```

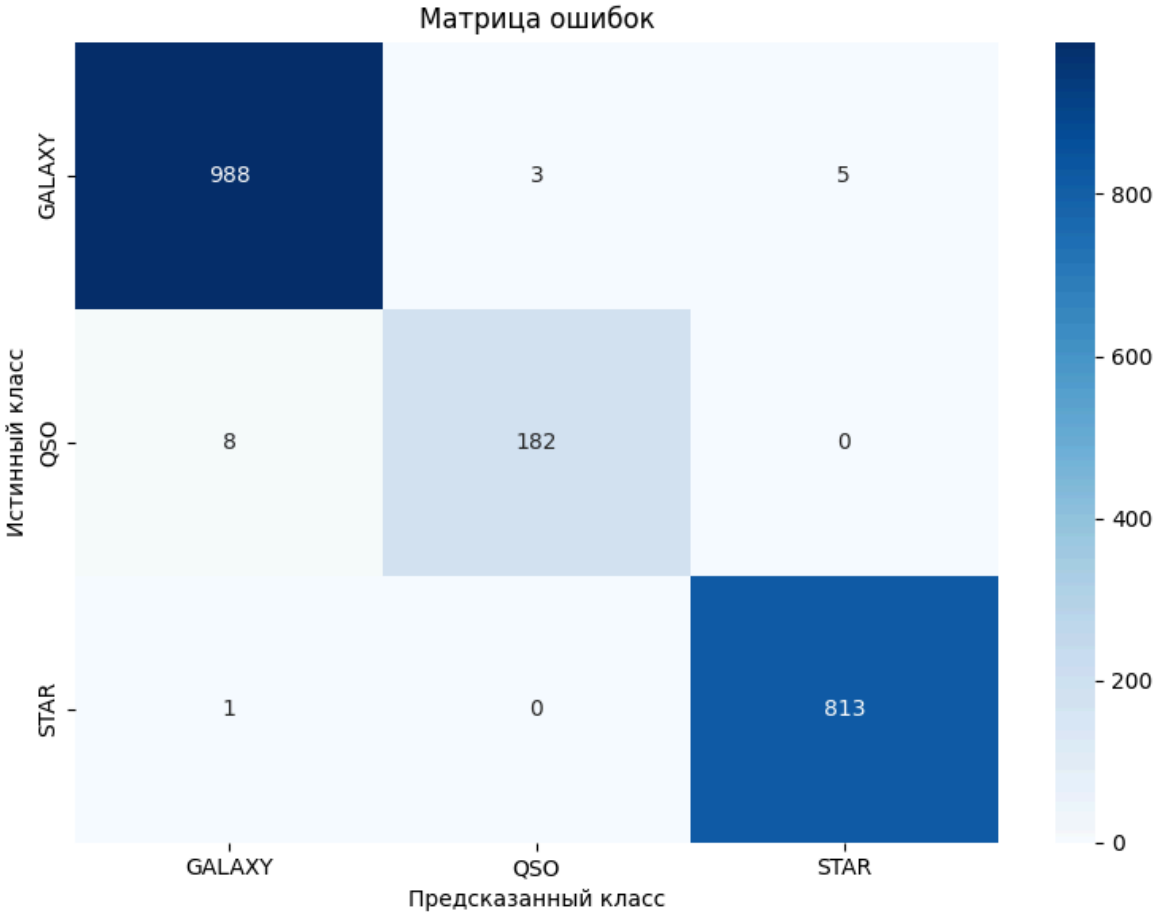
```
Результаты с стандартизацией:
1. Accuracy: 0.9900

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988978 0.990964  0.989970
   QSO   0.983607 0.947368  0.965147
  STAR   0.992674 0.998771  0.995713

3. Macro F1: 0.9836

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     987    3     6
QSO         10  180     0
STAR         1    0   813
```



Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [12]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

rf_grid = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf_grid, param_grid, cv=5, scoring='f1_macro',
grid_search.fit(X_train_class, y_train_class)

print(f"Лучшие параметры: {grid_search.best_params_}")
print(f"Лучший score на кросс-валидации: {grid_search.best_score_:.4f}")

rf_classifier_tuned = grid_search.best_estimator_
y_pred_class_tuned = rf_classifier_tuned.predict(X_test_class)

print("\nРезультаты с подобранными гиперпараметрами:")
metrics_tuned = evaluate_classification_model(y_test_class, y_pred_class_
```

```
Лучшие параметры: {'max_depth': 20, 'min_samples_split': 5, 'n_estimator
s': 100}
Лучший score на кросс-валидации: 0.9797

Результаты с подобранными гиперпараметрами:
1. Accuracy: 0.9915

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.990973 0.991968  0.991470
   QSO   0.983784 0.957895  0.970667
  STAR   0.993888 0.998771  0.996324

3. Macro F1: 0.9862

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     988    3     5
QSO          8  182     0
STAR         1    0   813
```



Сравним результаты всех вариантов

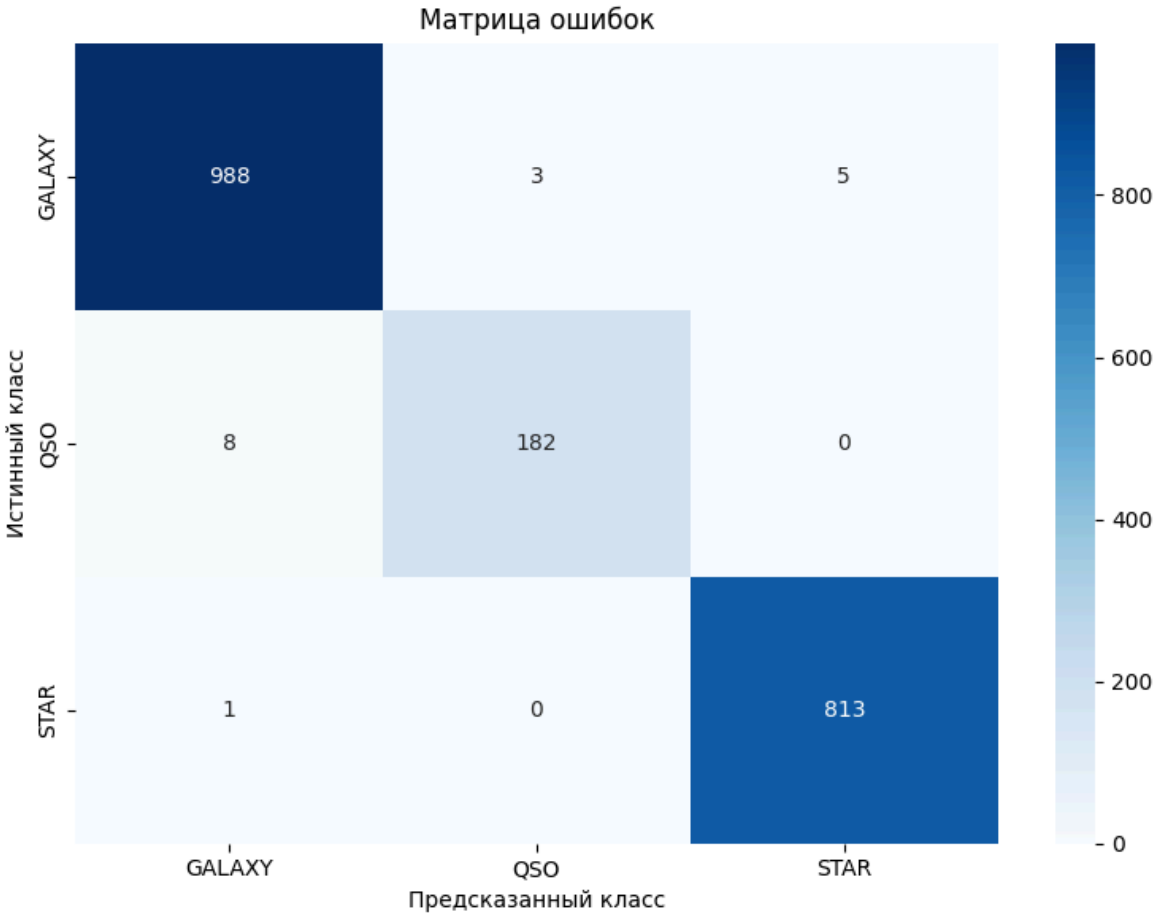
In [13]:
```
comparison_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Со стандартизацией', 'С подобранными
    'Accuracy': [class_metrics['accuracy'], metrics_scaled['accuracy'], m
    'Macro F1': [class_metrics['macro_f1'], metrics_scaled['macro_f1'], m
})

print("Сравнение моделей классификации:")
print(comparison_class.to_string(index=False))
```

```
Сравнение моделей классификации:
                         Модель  Accuracy  Macro F1
               Базовый бейзлайн    0.9900  0.983610
              Со стандартизацией    0.9900  0.983610
С подобранными гиперпараметрами    0.9915  0.986153
```

Сформируем улучшенный бейзлайн на основе лучших результатов. Используем подобранные гиперпараметры.

In [14]:
```
rf_classifier_improved = RandomForestClassifier(
    n_estimators=grid_search.best_params_['n_estimators'],
    max_depth=grid_search.best_params_['max_depth'],
    min_samples_split=grid_search.best_params_['min_samples_split'],
    random_state=42
)
rf_classifier_improved.fit(X_train_class, y_train_class)
y_pred_class_improved = rf_classifier_improved.predict(X_test_class)

print("Результаты улучшенного бейзлайна (классификация):")
class_metrics_improved = evaluate_classification_model(y_test_class, y_pr
```

```
Результаты улучшенного бейзлайна (классификация):
1. Accuracy: 0.9915

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.990973 0.991968  0.991470
   QSO   0.983784 0.957895  0.970667
  STAR   0.993888 0.998771  0.996324

3. Macro F1: 0.9862

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     988    3     5
QSO          8  182     0
STAR         1    0   813
```



Регрессия

Гипотеза 1: Стандартизация признаков улучшит качество модели.

In [15]:
```
scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)

rf_regressor_scaled = RandomForestRegressor(n_estimators=100, random_stat
rf_regressor_scaled.fit(X_train_reg_scaled, y_train_reg)
y_pred_reg_scaled = rf_regressor_scaled.predict(X_test_reg_scaled)

print("Результаты с стандартизацией:")
metrics_reg_scaled = evaluate_regression_model(y_test_reg, y_pred_reg_sca
```

```
Результаты с стандартизацией:
1. MAE: 1.5922

2. MSE: 5.0999

3. RMSE: 2.2583

4. R²: 0.5289
```

Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [16]:
```
param_grid_reg = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
```

```
    'min_samples_split': [2, 5, 10]
}

rf_grid_reg = RandomForestRegressor(random_state=42)
grid_search_reg = GridSearchCV(rf_grid_reg, param_grid_reg, cv=5, scoring
grid_search_reg.fit(X_train_reg, y_train_reg)

print(f"Лучшие параметры: {grid_search_reg.best_params_}")
print(f"Лучший score на кросс-валидации: {grid_search_reg.best_score_:.4f

rf_regressor_tuned = grid_search_reg.best_estimator_
y_pred_reg_tuned = rf_regressor_tuned.predict(X_test_reg)

print("\nРезультаты с подобранными гиперпараметрами:")
metrics_reg_tuned = evaluate_regression_model(y_test_reg, y_pred_reg_tune
```

```
Лучшие параметры: {'max_depth': 10, 'min_samples_split': 10, 'n_estimator
s': 200}
Лучший score на кросс-валидации: 0.5513

Результаты с подобранными гиперпараметрами:
1. MAE: 1.5539

2. MSE: 4.9293

3. RMSE: 2.2202

4. R²: 0.5446
```

Гипотеза 3: Формирование новых признаков улучшит качество модели.

In [17]:
```
X_train_reg_features = X_train_reg.copy()
X_test_reg_features = X_test_reg.copy()

X_train_reg_features['weight_ratio'] = X_train_reg['Whole weight'] / (X_t
X_test_reg_features['weight_ratio'] = X_test_reg['Whole weight'] / (X_tes

X_train_reg_features['volume_approx'] = X_train_reg['Length'] * X_train_r
X_test_reg_features['volume_approx'] = X_test_reg['Length'] * X_test_reg[

rf_regressor_features = RandomForestRegressor(n_estimators=100, random_st
rf_regressor_features.fit(X_train_reg_features, y_train_reg)
y_pred_reg_features = rf_regressor_features.predict(X_test_reg_features)

print("Результаты с новыми признаками:")
metrics_reg_features = evaluate_regression_model(y_test_reg, y_pred_reg_f
```

```
Результаты с новыми признаками:
1. MAE: 1.5552

2. MSE: 4.7994

3. RMSE: 2.1908

4. R²: 0.5566
```

Сравним результаты всех вариантов:

In [18]:
```
comparison_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Со стандартизацией', 'С подобранными
```

```
    'MAE': [reg_metrics['mae'], metrics_reg_scaled['mae'], metrics_reg_tu
    'RMSE': [reg_metrics['rmse'], metrics_reg_scaled['rmse'], metrics_reg
    'R²': [reg_metrics['r2'], metrics_reg_scaled['r2'], metrics_reg_tuned
})

print("Сравнение моделей регрессии:")
print(comparison_reg.to_string(index=False))
```

```
Сравнение моделей регрессии:
                        Модель      MAE     RMSE       R²
                Базовый бейзлайн 1.589928 2.256425 0.529667
            Со стандартизацией 1.592213 2.258295 0.528887
С подобранными гиперпараметрами 1.553872 2.220208 0.544644
            С новыми признаками 1.555203 2.190763 0.556642
```

Сформируем улучшенный бейзлайн на основе лучших результатов. Используем подобранные гиперпараметры и новые признаки.

In [19]:
```
rf_regressor_improved = RandomForestRegressor(
    n_estimators=grid_search_reg.best_params_['n_estimators'],
    max_depth=grid_search_reg.best_params_['max_depth'],
    min_samples_split=grid_search_reg.best_params_['min_samples_split'],
    random_state=42
)
rf_regressor_improved.fit(X_train_reg_features, y_train_reg)
y_pred_reg_improved = rf_regressor_improved.predict(X_test_reg_features)

print("Результаты улучшенного бейзлайна (регрессия):")
reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_reg_i
```

```
Результаты улучшенного бейзлайна (регрессия):
1. MAE: 1.5198

2. MSE: 4.6377

3. RMSE: 2.1535

4. R²: 0.5716
```

Сравним результаты улучшенного бейзлайна с базовым

In [20]:
```
print("Сравнение базового и улучшенного бейзлайна (классификация):")
comparison_class_final = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'Accuracy': [class_metrics['accuracy'], class_metrics_improved['accur
    'Macro F1': [class_metrics['macro_f1'], class_metrics_improved['macro
})
print(comparison_class_final.to_string(index=False))

print("\nСравнение базового и улучшенного бейзлайна (регрессия):")
comparison_reg_final = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'MAE': [reg_metrics['mae'], reg_metrics_improved['mae']],
    'RMSE': [reg_metrics['rmse'], reg_metrics_improved['rmse']],
    'R²': [reg_metrics['r2'], reg_metrics_improved['r2']]
})
print(comparison_reg_final.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна (классификация):
             Модель  Accuracy  Macro F1
   Базовый бейзлайн    0.9900  0.983610
Улучшенный бейзлайн    0.9915  0.986153

Сравнение базового и улучшенного бейзлайна (регрессия):
             Модель      MAE     RMSE       R²
   Базовый бейзлайн 1.589928 2.256425 0.529667
Улучшенный бейзлайн 1.519789 2.153528 0.571585
```

4. Имплементация алгоритма машинного обучения

Перейдем к имлпементации алгоритмов

Классификация

Реализуем алгоритм Random Forest для классификации

In [21]:
```python
import numpy as np
from collections import Counter

class DecisionTreeClassifier:
    def __init__(self, max_depth=10, min_samples_split=2, random_state=No
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.random_state = random_state
        self.tree = None

    def _gini_impurity(self, y):
        if len(y) == 0:
            return 0
        counts = Counter(y)
        impurity = 1
        for label in counts:
            prob = counts[label] / len(y)
            impurity -= prob ** 2
        return impurity

    def _split(self, X, y, feature_idx, threshold):
        left_mask = X[:, feature_idx] <= threshold
        right_mask = ~left_mask
        return left_mask, right_mask

    def _best_split(self, X, y, feature_indices):
        best_gini = float('inf')
        best_feature = None
        best_threshold = None

        for feature_idx in feature_indices:
            thresholds = np.unique(X[:, feature_idx])
            for threshold in thresholds:
                left_mask, right_mask = self._split(X, y, feature_idx, th

                if np.sum(left_mask) == 0 or np.sum(right_mask) == 0:
                    continue
```

```
                left_gini = self._gini_impurity(y[left_mask])
                right_gini = self._gini_impurity(y[right_mask])

                weighted_gini = (np.sum(left_mask) * left_gini +
                                 np.sum(right_mask) * right_gini) / len(y

                if weighted_gini < best_gini:
                    best_gini = weighted_gini
                    best_feature = feature_idx
                    best_threshold = threshold

        return best_feature, best_threshold

    def _build_tree(self, X, y, depth, feature_indices):
        if depth >= self.max_depth or len(y) < self.min_samples_split:
            return Counter(y).most_common(1)[0][0]

        if len(np.unique(y)) == 1:
            return y[0]

        best_feature, best_threshold = self._best_split(X, y, feature_ind

        if best_feature is None:
            return Counter(y).most_common(1)[0][0]

        left_mask, right_mask = self._split(X, y, best_feature, best_thre

        tree = {
            'feature': best_feature,
            'threshold': best_threshold,
            'left': self._build_tree(X[left_mask], y[left_mask], depth +
            'right': self._build_tree(X[right_mask], y[right_mask], depth
        }

        return tree

    def _predict_sample(self, x, tree):
        if isinstance(tree, dict):
            if x[tree['feature']] <= tree['threshold']:
                return self._predict_sample(x, tree['left'])
            else:
                return self._predict_sample(x, tree['right'])
        else:
            return tree

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        feature_indices = list(range(X.shape[1]))
        self.tree = self._build_tree(X, y, 0, feature_indices)
        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []
        for x in X:
            predictions.append(self._predict_sample(x, self.tree))
        return np.array(predictions)
```

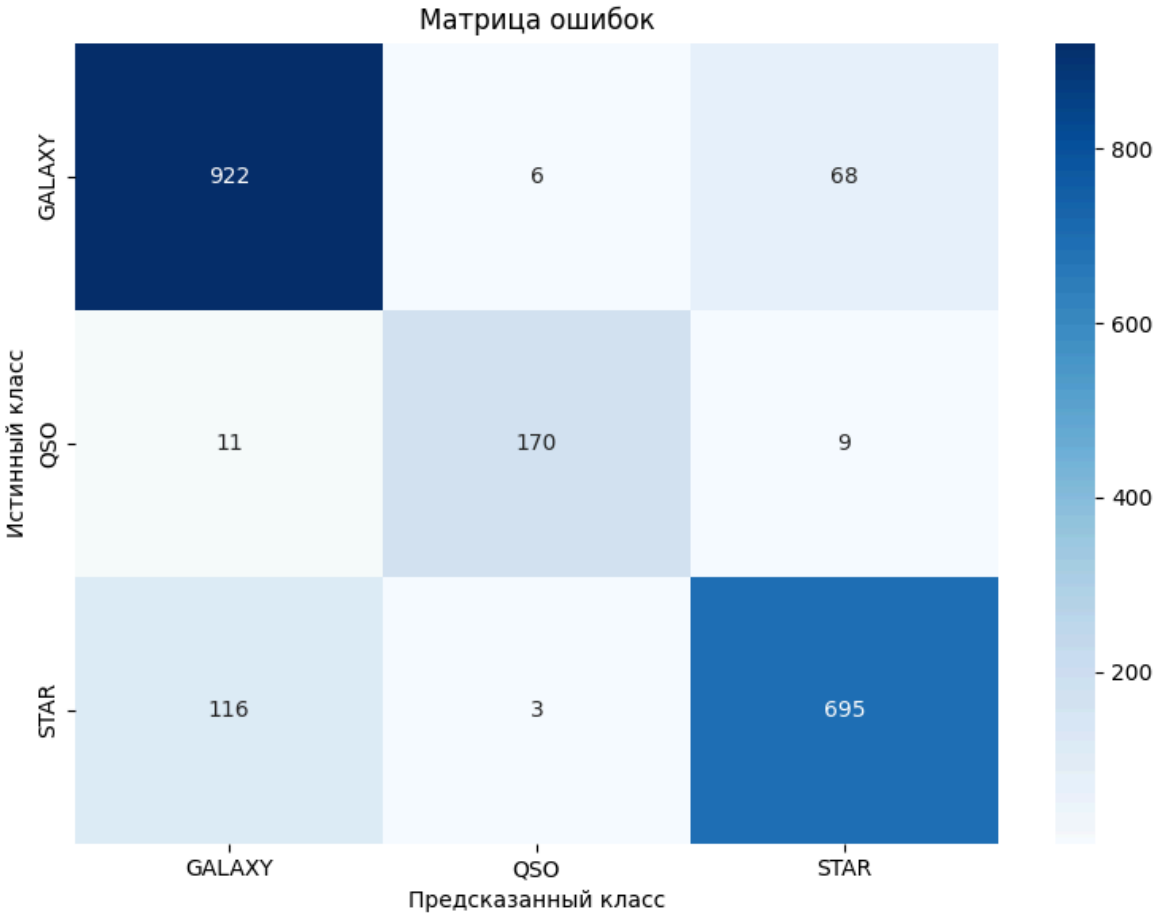
```python
class MyRandomForestClassifier:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=
                 max_features='sqrt', random_state=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.random_state = random_state
        self.trees = []
        self.feature_indices_list = []

    def _bootstrap_sample(self, X, y):
        n_samples = X.shape[0]
        indices = np.random.choice(n_samples, n_samples, replace=True)
        return X[indices], y[indices]

    def _get_feature_indices(self, n_features):
        if self.max_features == 'sqrt':
            n_selected = int(np.sqrt(n_features))
        elif self.max_features == 'log2':
            n_selected = int(np.log2(n_features))
        else:
            n_selected = self.max_features

        return np.random.choice(n_features, n_selected, replace=False)

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        n_features = X.shape[1]

        if self.random_state is not None:
            np.random.seed(self.random_state)

        for i in range(self.n_estimators):
            X_boot, y_boot = self._bootstrap_sample(X, y)

            feature_indices = self._get_feature_indices(n_features)
            self.feature_indices_list.append(feature_indices)

            tree = DecisionTreeClassifier(
                max_depth=self.max_depth,
                min_samples_split=self.min_samples_split,
                random_state=self.random_state
            )
            tree.fit(X_boot[:, feature_indices], y_boot)
            self.trees.append(tree)

        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []

        for x in X:
            tree_predictions = []
            for i, tree in enumerate(self.trees):
                feature_indices = self.feature_indices_list[i]
                tree_pred = tree._predict_sample(x[feature_indices], tree
                tree_predictions.append(tree_pred)
```

```
            predictions.append(Counter(tree_predictions).most_common(1)[0

        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [22]:
```
my_rf_classifier = MyRandomForestClassifier(n_estimators=50, max_depth=10
                                            min_samples_split=2, random_s
my_rf_classifier.fit(X_train_class.values, y_train_class)
y_pred_my_class = my_rf_classifier.predict(X_test_class.values)

print("Результаты имплементированной модели классификации:")
my_class_metrics = evaluate_classification_model(y_test_class, y_pred_my_
```

```
Результаты имплементированной модели классификации:
1. Accuracy: 0.8935

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.878932 0.925703  0.901711
   QSO   0.949721 0.894737  0.921409
  STAR   0.900259 0.853808  0.876419

3. Macro F1: 0.8998

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     922    6    68
QSO         11  170     9
STAR       116    3   695
```



Сравним результаты имплементированной модели с базовым бейзлайном

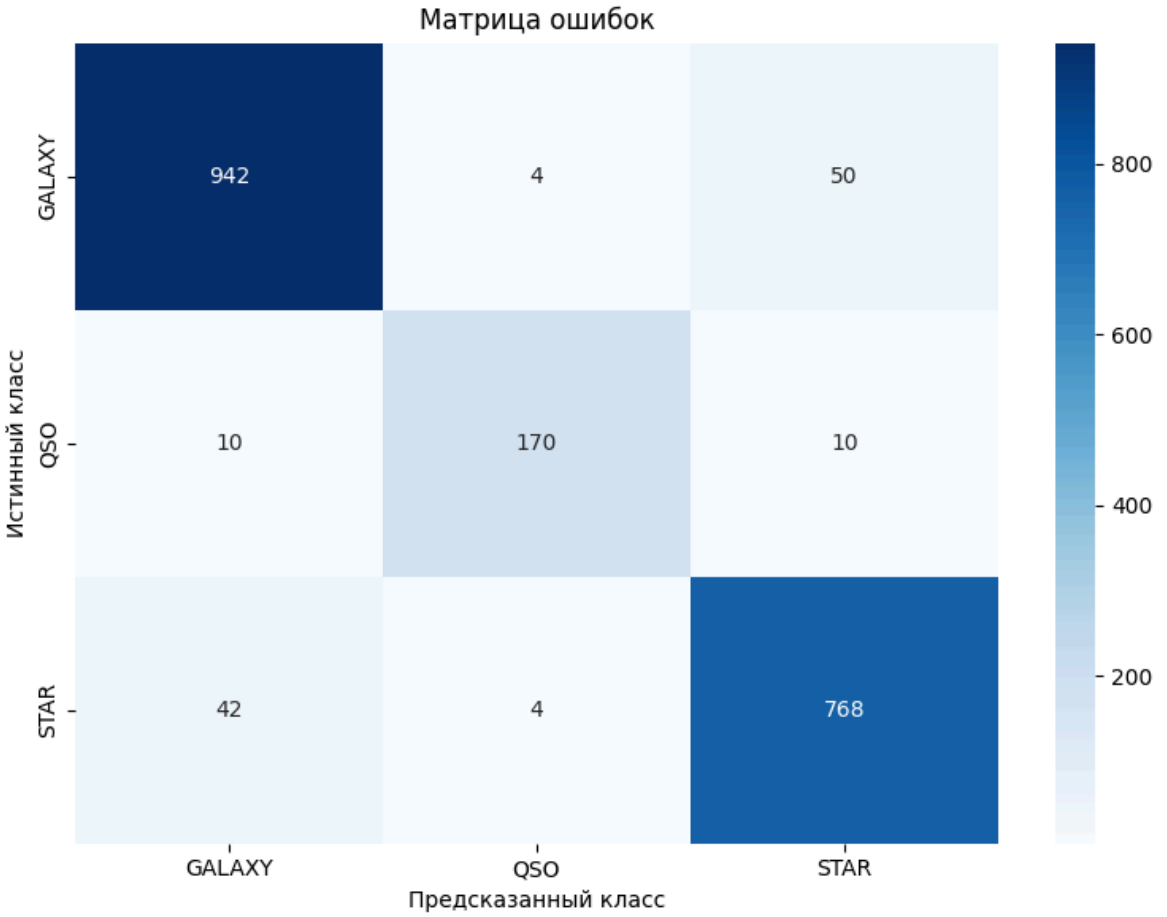
In [23]:
```
comparison_my_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель']
    'Accuracy': [class_metrics['accuracy'], my_class_metrics['accuracy']]
    'Macro F1': [class_metrics['macro_f1'], my_class_metrics['macro_f1']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_class.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                  Модель  Accuracy  Macro F1
Базовый бейзлайн (sklearn)    0.9900  0.983610
 Имплементированная модель    0.8935  0.899846
```

Теперь применим техники из улучшенного бейзлайна

In [24]:
```
best_n_estimators = grid_search.best_params_['n_estimators']
best_max_depth = grid_search.best_params_['max_depth'] if grid_search.bes
best_min_samples_split = grid_search.best_params_['min_samples_split']

my_rf_classifier_improved = MyRandomForestClassifier(
    n_estimators=min(best_n_estimators, 50),
    max_depth=best_max_depth,
    min_samples_split=best_min_samples_split,
    random_state=42
)
my_rf_classifier_improved.fit(X_train_class.values, y_train_class)
y_pred_my_class_improved = my_rf_classifier_improved.predict(X_test_class

print("Результаты имплементированной модели с улучшениями:")
my_class_metrics_improved = evaluate_classification_model(y_test_class, y
```

```
Результаты имплементированной модели с улучшениями:
1. Accuracy: 0.9400

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.947686 0.945783  0.946734
   QSO   0.955056 0.894737  0.923913
  STAR   0.927536 0.943489  0.935445

3. Macro F1: 0.9354

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     942    4    50
QSO         10  170    10
STAR        42    4   768
```



Сравним результаты имплементированной модели с улучшенным бейзлайном

In [25]:
```
comparison_my_class_final = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модел
    'Accuracy': [class_metrics_improved['accuracy'], my_class_metrics_imp
    'Macro F1': [class_metrics_improved['macro_f1'], my_class_metrics_imp
})

print("Сравнение имплементированной модели с улучшенным бейзлайном:")
print(comparison_my_class_final.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшенным бейзлайном:
                                Модель  Accuracy  Macro F1
          Улучшенный бейзлайн (sklearn)    0.9915  0.986153
Имплементированная модель с улучшениями    0.9400  0.935364
```

Регрессия

Реализуем алгоритм Random Forest для регрессии

In [26]:
```
class DecisionTreeRegressor:
    def __init__(self, max_depth=10, min_samples_split=2, random_state=No
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.random_state = random_state
        self.tree = None

    def _mse(self, y):
        if len(y) == 0:
            return 0
```

```
        return np.mean((y - np.mean(y)) ** 2)

    def _split(self, X, y, feature_idx, threshold):
        left_mask = X[:, feature_idx] <= threshold
        right_mask = ~left_mask
        return left_mask, right_mask

    def _best_split(self, X, y, feature_indices):
        best_mse = float('inf')
        best_feature = None
        best_threshold = None

        for feature_idx in feature_indices:
            thresholds = np.unique(X[:, feature_idx])
            for threshold in thresholds:
                left_mask, right_mask = self._split(X, y, feature_idx, th

                if np.sum(left_mask) == 0 or np.sum(right_mask) == 0:
                    continue

                left_mse = self._mse(y[left_mask])
                right_mse = self._mse(y[right_mask])

                weighted_mse = (np.sum(left_mask) * left_mse +
                                np.sum(right_mask) * right_mse) / len(y)

                if weighted_mse < best_mse:
                    best_mse = weighted_mse
                    best_feature = feature_idx
                    best_threshold = threshold

        return best_feature, best_threshold

    def _build_tree(self, X, y, depth, feature_indices):
        if depth >= self.max_depth or len(y) < self.min_samples_split:
            return np.mean(y)

        if len(np.unique(y)) == 1:
            return y[0]

        best_feature, best_threshold = self._best_split(X, y, feature_ind

        if best_feature is None:
            return np.mean(y)

        left_mask, right_mask = self._split(X, y, best_feature, best_thre

        tree = {
            'feature': best_feature,
            'threshold': best_threshold,
            'left': self._build_tree(X[left_mask], y[left_mask], depth +
            'right': self._build_tree(X[right_mask], y[right_mask], depth
        }

        return tree

    def _predict_sample(self, x, tree):
        if isinstance(tree, dict):
            if x[tree['feature']] <= tree['threshold']:
                return self._predict_sample(x, tree['left'])
```

```
            else:
                return self._predict_sample(x, tree['right'])
        else:
            return tree

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        feature_indices = list(range(X.shape[1]))
        self.tree = self._build_tree(X, y, 0, feature_indices)
        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []
        for x in X:
            predictions.append(self._predict_sample(x, self.tree))
        return np.array(predictions)


class MyRandomForestRegressor:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=
                 max_features='sqrt', random_state=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.random_state = random_state
        self.trees = []
        self.feature_indices_list = []

    def _bootstrap_sample(self, X, y):
        n_samples = X.shape[0]
        indices = np.random.choice(n_samples, n_samples, replace=True)
        return X[indices], y[indices]

    def _get_feature_indices(self, n_features):
        if self.max_features == 'sqrt':
            n_selected = int(np.sqrt(n_features))
        elif self.max_features == 'log2':
            n_selected = int(np.log2(n_features))
        else:
            n_selected = self.max_features

        return np.random.choice(n_features, n_selected, replace=False)

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        n_features = X.shape[1]

        if self.random_state is not None:
            np.random.seed(self.random_state)

        for i in range(self.n_estimators):
            X_boot, y_boot = self._bootstrap_sample(X, y)

            feature_indices = self._get_feature_indices(n_features)
            self.feature_indices_list.append(feature_indices)
```

```
            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
                min_samples_split=self.min_samples_split,
                random_state=self.random_state
            )
            tree.fit(X_boot[:, feature_indices], y_boot)
            self.trees.append(tree)

        return self

    def predict(self, X):
        X = np.array(X)
        predictions = []

        for x in X:
            tree_predictions = []
            for i, tree in enumerate(self.trees):
                feature_indices = self.feature_indices_list[i]
                tree_pred = tree._predict_sample(x[feature_indices], tree
                tree_predictions.append(tree_pred)

            predictions.append(np.mean(tree_predictions))

        return np.array(predictions)
```

Обучим имплементированную модель на исходных данных

In [27]:
```
my_rf_regressor = MyRandomForestRegressor(n_estimators=50, max_depth=10,
                                          min_samples_split=2, random_stat
my_rf_regressor.fit(X_train_reg.values, y_train_reg.values)
y_pred_my_reg = my_rf_regressor.predict(X_test_reg.values)

print("Результаты имплементированной модели регрессии:")
my_reg_metrics = evaluate_regression_model(y_test_reg, y_pred_my_reg)
```

```
Результаты имплементированной модели регрессии:
1. MAE: 1.7823

2. MSE: 6.2573

3. RMSE: 2.5015

4. R²: 0.4220
```

Сравним результаты имплементированной модели с базовым бейзлайном

In [28]:
```
comparison_my_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель']
    'MAE': [reg_metrics['mae'], my_reg_metrics['mae']],
    'RMSE': [reg_metrics['rmse'], my_reg_metrics['rmse']],
    'R²': [reg_metrics['r2'], my_reg_metrics['r2']]
})

print("Сравнение имплементированной модели с базовым бейзлайном:")
print(comparison_my_reg.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном:
                    Модель      MAE     RMSE       R²
Базовый бейзлайн (sklearn) 1.589928 2.256425 0.529667
 Имплементированная модель 1.782342 2.501466 0.421967
```

Теперь применим техники из улучшенного бейзлайна

In [29]:
```
best_n_estimators_reg = grid_search_reg.best_params_['n_estimators']
best_max_depth_reg = grid_search_reg.best_params_['max_depth'] if grid_se
best_min_samples_split_reg = grid_search_reg.best_params_['min_samples_sp

my_rf_regressor_improved = MyRandomForestRegressor(
    n_estimators=min(best_n_estimators_reg, 50),
    max_depth=best_max_depth_reg,
    min_samples_split=best_min_samples_split_reg,
    random_state=42
)
my_rf_regressor_improved.fit(X_train_reg_features.values, y_train_reg.val
y_pred_my_reg_improved = my_rf_regressor_improved.predict(X_test_reg_feat

print("Результаты имплементированной модели с улучшениями:")
my_reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_my
```

```
Результаты имплементированной модели с улучшениями:
1. MAE: 1.6099

2. MSE: 5.1879

3. RMSE: 2.2777

4. R²: 0.5208
```

Сравним результаты имплементированной модели с улучшенным бейзлайном

In [30]:
```
comparison_my_reg_final = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модел
    'MAE': [reg_metrics_improved['mae'], my_reg_metrics_improved['mae']],
    'RMSE': [reg_metrics_improved['rmse'], my_reg_metrics_improved['rmse'
    'R²': [reg_metrics_improved['r2'], my_reg_metrics_improved['r2']]
})

print("Сравнение имплементированной модели с улучшенным бейзлайном:")
print(comparison_my_reg_final.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшенным бейзлайном:
                              Модель      MAE     RMSE       R²
          Улучшенный бейзлайн (sklearn) 1.519789 2.153528 0.571585
Имплементированная модель с улучшениями 1.609885 2.277700 0.520756
```

Общие выводы по результатам всех моделей

Сравним все 4 модели для классификации и регрессии: базовый бейзлайн из sklearn, имплементированную модель базового бейзлайна, модель с улучшенным бейзлайном из sklearn и имплементированную модель улучшенного бейзлайна.

Классификация

In [31]:
```
final_comparison_class = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'Accuracy': [
        class_metrics['accuracy'],
        my_class_metrics['accuracy'],
        class_metrics_improved['accuracy'],
        my_class_metrics_improved['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        my_class_metrics['macro_f1'],
        class_metrics_improved['macro_f1'],
        my_class_metrics_improved['macro_f1']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ")
print(final_comparison_class.to_string(index=False))

print("\nВЫВОДЫ ПО КЛАССИФИКАЦИИ:")
print(f"1. Базовый бейзлайн (sklearn): Accuracy = {class_metrics['accurac
print(f"2. Имплементированная модель базового бейзлайна: Accuracy = {my_c
print(f"3. Улучшенный бейзлайн (sklearn): Accuracy = {class_metrics_impro
print(f"4. Имплементированная модель улучшенного бейзлайна: Accuracy = {m
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sk
print(f"  Accuracy: {((class_metrics_improved['accuracy'] - class_metrics
print(f"  Macro F1: {((class_metrics_improved['macro_f1'] - class_metrics
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзла
print(f"  Разница в Accuracy: {abs(class_metrics_improved['accuracy'] - m
print(f"  Разница в Macro F1: {abs(class_metrics_improved['macro_f1'] - m
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ
                                      Модель  Accuracy  Macro F1
                        Базовый бейзлайн (sklearn)    0.9900  0.983610
   Имплементированная модель базового бейзлайна    0.8935  0.899846
                    Улучшенный бейзлайн (sklearn)    0.9915  0.986153
Имплементированная модель улучшенного бейзлайна    0.9400  0.935364

ВЫВОДЫ ПО КЛАССИФИКАЦИИ:
1. Базовый бейзлайн (sklearn): Accuracy = 0.9900, Macro F1 = 0.9836
2. Имплементированная модель базового бейзлайна: Accuracy = 0.8935, Macro
F1 = 0.8998
3. Улучшенный бейзлайн (sklearn): Accuracy = 0.9915, Macro F1 = 0.9862
4. Имплементированная модель улучшенного бейзлайна: Accuracy = 0.9400, Mac
ro F1 = 0.9354

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  Accuracy: 0.15%
  Macro F1: 0.26%

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в Accuracy: 0.051500
  Разница в Macro F1: 0.050790
```

Регрессия

In [32]:
```
final_comparison_reg = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'MAE': [
        reg_metrics['mae'],
        my_reg_metrics['mae'],
        reg_metrics_improved['mae'],
        my_reg_metrics_improved['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        my_reg_metrics['rmse'],
        reg_metrics_improved['rmse'],
        my_reg_metrics_improved['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        my_reg_metrics['r2'],
        reg_metrics_improved['r2'],
        my_reg_metrics_improved['r2']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ")
print(final_comparison_reg.to_string(index=False))

print("\nВЫВОДЫ ПО РЕГРЕССИИ:")
print(f"1. Базовый бейзлайн (sklearn): MAE = {reg_metrics['mae']:.4f}, RM
print(f"2. Имплементированная модель базового бейзлайна: MAE = {my_reg_me
print(f"3. Улучшенный бейзлайн (sklearn): MAE = {reg_metrics_improved['ma
print(f"4. Имплементированная модель улучшенного бейзлайна: MAE = {my_reg
print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sk
print(f"  MAE: {((reg_metrics['mae'] - reg_metrics_improved['mae']) / reg
print(f"  RMSE: {((reg_metrics['rmse'] - reg_metrics_improved['rmse']) /
print(f"  R²: {((reg_metrics_improved['r2'] - reg_metrics['r2']) / abs(re
print("\nСравнение имплементированной модели с sklearn (улучшенный бейзла
print(f"  Разница в MAE: {abs(reg_metrics_improved['mae'] - my_reg_metric
print(f"  Разница в RMSE: {abs(reg_metrics_improved['rmse'] - my_reg_metr
print(f"  Разница в R²: {abs(reg_metrics_improved['r2'] - my_reg_metrics_
```

ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ

Модель	MAE	RMSE	R^2
Базовый бейзлайн (sklearn)	1.589928	2.256425	0.529667
Имплементированная модель базового бейзлайна	1.782342	2.501466	0.421967
Улучшенный бейзлайн (sklearn)	1.519789	2.153528	0.571585
Имплементированная модель улучшенного бейзлайна	1.609885	2.277700	0.520756

ВЫВОДЫ ПО РЕГРЕССИИ:
1. Базовый бейзлайн (sklearn): MAE = 1.5899, RMSE = 2.2564, R^2 = 0.5297
2. Имплементированная модель базового бейзлайна: MAE = 1.7823, RMSE = 2.5015, R^2 = 0.4220
3. Улучшенный бейзлайн (sklearn): MAE = 1.5198, RMSE = 2.1535, R^2 = 0.5716
4. Имплементированная модель улучшенного бейзлайна: MAE = 1.6099, RMSE = 2.2777, R^2 = 0.5208

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  MAE: 4.41% улучшение
  RMSE: 4.56% улучшение
  R^2: 7.91% улучшение

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в MAE: 0.090095
  Разница в RMSE: 0.124173
  Разница в R^2: 0.050829

Лабораторная работа №5 (Проведение исследований с градиентным бустингом)

2. Создание бейзлайна и оценка качества

Перейдем к созданию базовых моделей

Классификация

Загрузим датасет и посмотрим на данные

In [4]:
```
import pandas as pd

df_class = pd.read_csv('data/Skyserver_SQL2_27_2018 6_51_39 PM.csv')

print(f"Размерность данных")
print(df_class.shape)
print(f"\nИнформация о данных")
print(df_class.info())
print(f"\nПервые 5 строк")
print(df_class.head())
print(f"\nРаспределение классов")
print(df_class['class'].value_counts())
```

```
Размерность данных
(10000, 18)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   objid      10000 non-null  float64
 1   ra         10000 non-null  float64
 2   dec        10000 non-null  float64
 3   u          10000 non-null  float64
 4   g          10000 non-null  float64
 5   r          10000 non-null  float64
 6   i          10000 non-null  float64
 7   z          10000 non-null  float64
 8   run        10000 non-null  int64  
 9   rerun      10000 non-null  int64  
 10  camcol     10000 non-null  int64  
 11  field      10000 non-null  int64  
 12  specobjid  10000 non-null  float64
 13  class      10000 non-null  object 
 14  redshift   10000 non-null  float64
 15  plate      10000 non-null  int64  
 16  mjd        10000 non-null  int64  
 17  fiberid    10000 non-null  int64  
dtypes: float64(10), int64(7), object(1)
memory usage: 1.4+ MB
None

Первые 5 строк
          objid          ra       dec         u         g         r         i  \
0  1.237650e+18  183.531326  0.089693  19.47406  17.04240  15.94699  15.50342   
1  1.237650e+18  183.598370  0.135285  18.66280  17.21449  16.67637  16.48922   
2  1.237650e+18  183.680207  0.126185  19.38298  18.19169  17.47428  17.08732   
3  1.237650e+18  183.870529  0.049911  17.76536  16.60272  16.16116  15.98233   
4  1.237650e+18  183.883288  0.102557  17.55025  16.26342  16.43869  16.55492   

          z  run  rerun  camcol  field     specobjid   class  redshift  plate  \
0  15.22531  752    301       4    267  3.722360e+18    STAR -0.000009   3306   
1  16.39150  752    301       4    267  3.638140e+17    STAR -0.000055    323   
2  16.80125  752    301       4    268  3.232740e+17  GALAXY  0.123111    287   
3  15.90438  752    301       4    269  3.722370e+18    STAR -0.000111   3306   
4  16.61326  752    301       4    269  3.722370e+18    STAR  0.000590   3306   

     mjd  fiberid  
0  54922      491  
1  51615      541  
2  52023      513  
3  54922      510  
4  54922      512  

Распределение классов
class
GALAXY    4998
STAR      4152
QSO        850
Name: count, dtype: int64
```

Выделим исходные признаки, которые непосредственно описывают физические свойства объектов и целевую переменную. А также закодируем таргет, так как это категориальный признак.

In [5]:
```python
from sklearn.preprocessing import LabelEncoder

X_class = df_class[['u', 'g', 'r', 'i', 'z', 'redshift']]
y_class = df_class['class']

le = LabelEncoder()
y_class_encoded = le.fit_transform(y_class)
class_names = le.classes_
```

Разделим данные на выборку для обучения и тестовую выборку.

In [6]:
```python
from sklearn.model_selection import train_test_split

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class_encoded, test_size=0.2, random_state=42
)
```

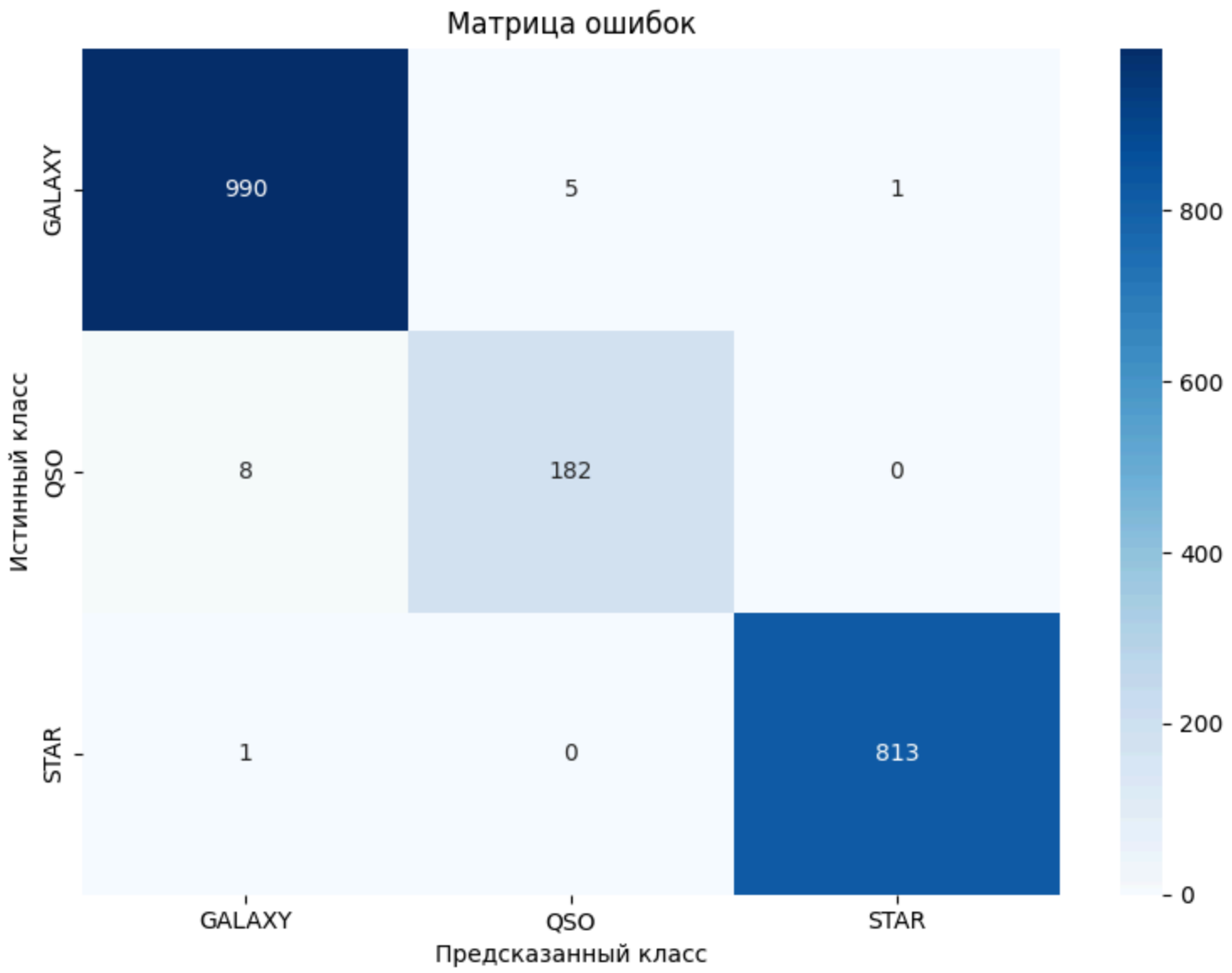
Обучим базовую модель GradientBoostingClassifier и выполним предсказания на тестовой выборке.

In [7]:
```python
from sklearn.ensemble import GradientBoostingClassifier

gb_classifier = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_classifier.fit(X_train_class, y_train_class)

y_pred_class = gb_classifier.predict(X_test_class)
```

Опишем функцию, которая будет использоваться для оценки обученной модели классификации.

In [8]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix
)

def evaluate_classification_model(y_true, y_pred, class_names):
    accuracy = accuracy_score(y_true, y_pred)
    print(f"1. Accuracy: {accuracy:.4f}")

    print(f"\n2. Метрики по классам:")
    precision = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)

    metrics_df = pd.DataFrame({
        'Класс': class_names,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
    print(metrics_df.to_string(index=False))

    macro_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"\n3. Macro F1: {macro_f1:.4f}")

    print(f"\n4. Матрица ошибок:")
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
    print(cm_df)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Матрица ошибок')
    plt.ylabel('Истинный класс')
    plt.xlabel('Предсказанный класс')
    plt.tight_layout()
    plt.show()

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'macro_f1': macro_f1,
        'confusion_matrix': cm
    }

class_metrics = evaluate_classification_model(y_test_class, y_pred_class, class_names)
```

```
1. Accuracy: 0.9925

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.990991 0.993976  0.992481
   QSO   0.973262 0.957895  0.965517
  STAR   0.998771 0.998771  0.998771

3. Macro F1: 0.9856

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     990    5     1
QSO          8  182     0
STAR         1    0   813
```



Регрессия

Загрузим датасет и посмотрим на данные

In [9]:
```
df_reg = pd.read_csv('data/abalone.csv')

print(f"Размерность данных")
print(df_reg.shape)
print(f"\nИнформация о данных")
print(df_reg.info())
print(f"\nПервые 5 строк")
print(df_reg.head())
print(f"\nСтатистика по числовым признакам")
print(df_reg.describe())
```

```
Размерность данных
(4177, 9)

Информация о данных
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None

Первые 5 строк
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7

Статистика по числовым признакам
            Length     Diameter       Height  Whole weight  Shucked weight  \
count  4177.000000  4177.000000  4177.000000   4177.000000     4177.000000
mean      0.523992     0.407881     0.139516      0.828742        0.359367
std       0.120093     0.099240     0.041827      0.490389        0.221963
min       0.075000     0.055000     0.000000      0.002000        0.001000
25%       0.450000     0.350000     0.115000      0.441500        0.186000
50%       0.545000     0.425000     0.140000      0.799500        0.336000
75%       0.615000     0.480000     0.165000      1.153000        0.502000
max       0.815000     0.650000     1.130000      2.825500        1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Выделим признаки и целевую переменную. Закодируем категориальный признак Sex.

In [10]:
```
X_reg = df_reg.drop('Rings', axis=1)
y_reg = df_reg['Rings']

le_sex = LabelEncoder()
X_reg_encoded = X_reg.copy()
X_reg_encoded['Sex'] = le_sex.fit_transform(X_reg['Sex'])
```

Разделим данные на выборку для обучения и тестовую выборку.

In [11]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg_encoded, y_reg, test_size=0.2, random_state=42
)
```

Обучим базовую модель GradientBoostingRegressor и выполним предсказания на тестовой выборке.

In [12]:
```
from sklearn.ensemble import GradientBoostingRegressor

gb_regressor = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = gb_regressor.predict(X_test_reg)
```

Опишем функцию, которая будет использоваться для оценки обученной модели регрессии.

In [13]:
```
import numpy as np
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
```

```
    r2_score,
)

def evaluate_regression_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    print(f"1. MAE: {mae:.4f}")

    mse = mean_squared_error(y_true, y_pred)
    print(f"\n2. MSE: {mse:.4f}")

    rmse = np.sqrt(mse)
    print(f"\n3. RMSE: {rmse:.4f}")

    r2 = r2_score(y_true, y_pred)
    print(f"\n4. R²: {r2:.4f}")

    return {
        'mae': mae,
        'mse': mse,
        'rmse': rmse,
        'r2': r2
    }

reg_metrics = evaluate_regression_model(y_test_reg, y_pred_reg)
```

```
1. MAE: 1.5740

2. MSE: 5.0375

3. RMSE: 2.2444

4. R²: 0.5347
```

3. Улучшение бейзлайна

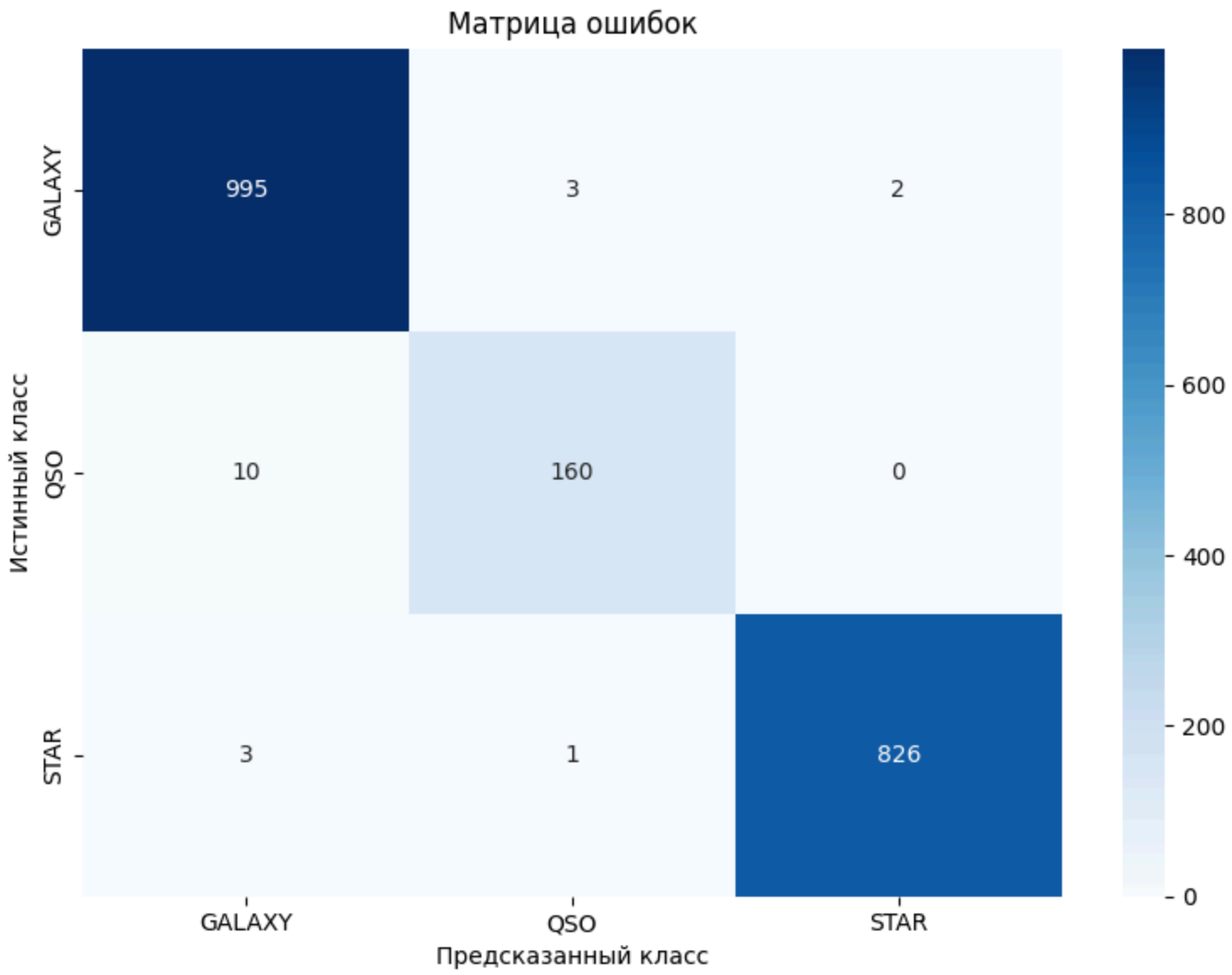
Перейдем к формулированию и проверкам гипотез

Классификация

Гипотеза 1: Добавление стандартизации признаков и стратификации при разбиении на выборки улучшит качество модели.

In [14]:
```
from sklearn.preprocessing import StandardScaler

X_train_class_h1, X_test_class_h1, y_train_class_h1, y_test_class_h1 = train_test_split(
    X_class, y_class_encoded, test_size=0.2, random_state=42, stratify=y_class_encoded
)

scaler_class_h1 = StandardScaler()
X_train_class_h1_scaled = scaler_class_h1.fit_transform(X_train_class_h1)
X_test_class_h1_scaled = scaler_class_h1.transform(X_test_class_h1)

gb_classifier_h1 = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_classifier_h1.fit(X_train_class_h1_scaled, y_train_class_h1)
y_pred_class_h1 = gb_classifier_h1.predict(X_test_class_h1_scaled)

print("Результаты гипотезы 1:")
class_metrics_h1 = evaluate_classification_model(y_test_class_h1, y_pred_class_h1, class_names)
```

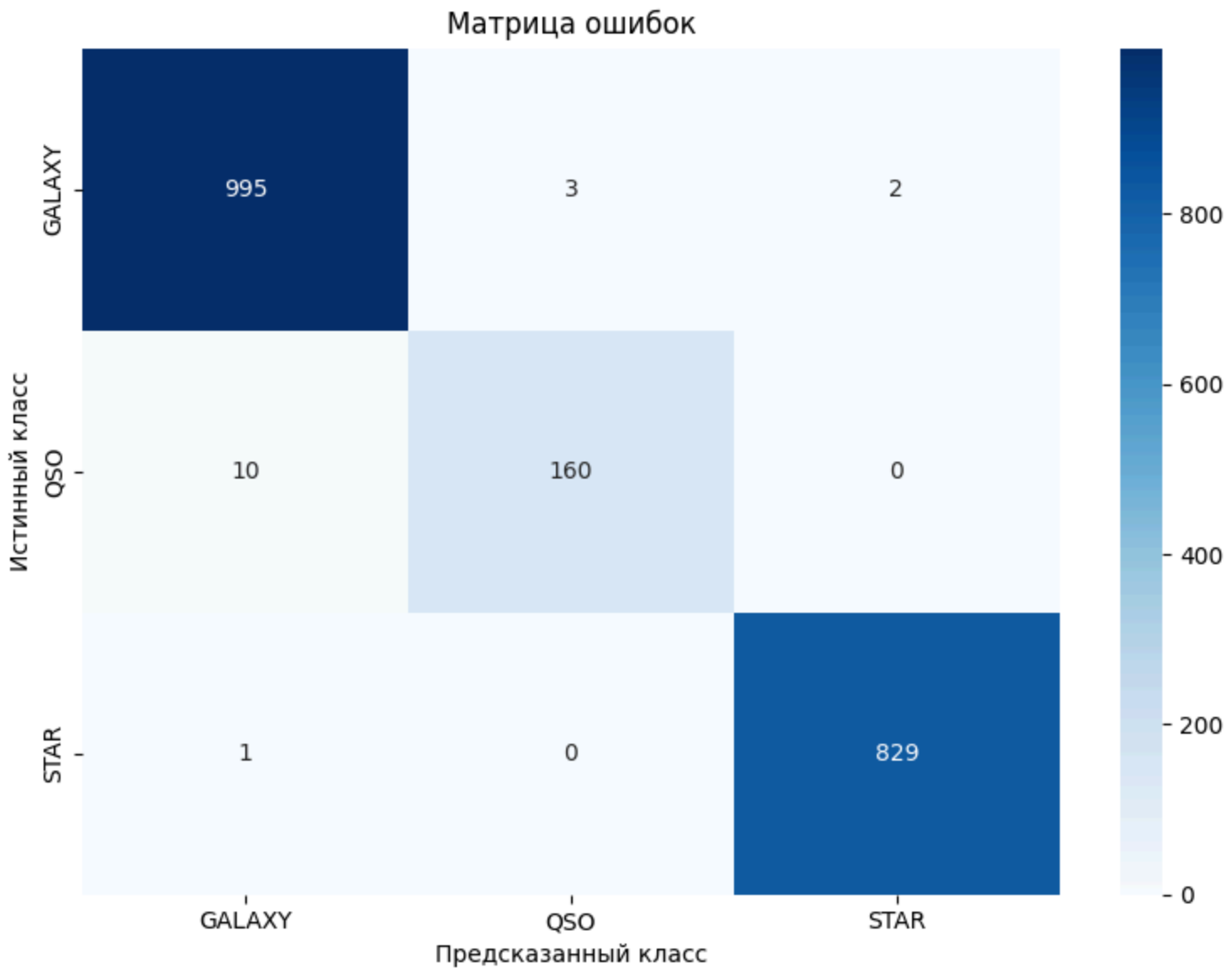
```
Результаты гипотезы 1:
1. Accuracy: 0.9905

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.987103 0.995000  0.991036
   QSO   0.975610 0.941176  0.958084
  STAR   0.997585 0.995181  0.996381

3. Macro F1: 0.9818

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     995    3     2
QSO         10  160     0
STAR         3    1   826
```



Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [15]:
```
from sklearn.model_selection import GridSearchCV

param_grid_h2 = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5]
}

gb_classifier_h2 = GradientBoostingClassifier(random_state=42)
grid_search_h2 = GridSearchCV(gb_classifier_h2, param_grid_h2, cv=5, scoring='f1_macro', n_jobs=-1)
grid_search_h2.fit(X_train_class_h1_scaled, y_train_class_h1)

print(f"Лучшие параметры: {grid_search_h2.best_params_}")
print(f"Лучший score на кросс-валидации: {grid_search_h2.best_score_:.4f}")

y_pred_class_h2 = grid_search_h2.predict(X_test_class_h1_scaled)

print("\nРезультаты гипотезы 2:")
class_metrics_h2 = evaluate_classification_model(y_test_class_h1, y_pred_class_h2, class_names)
```

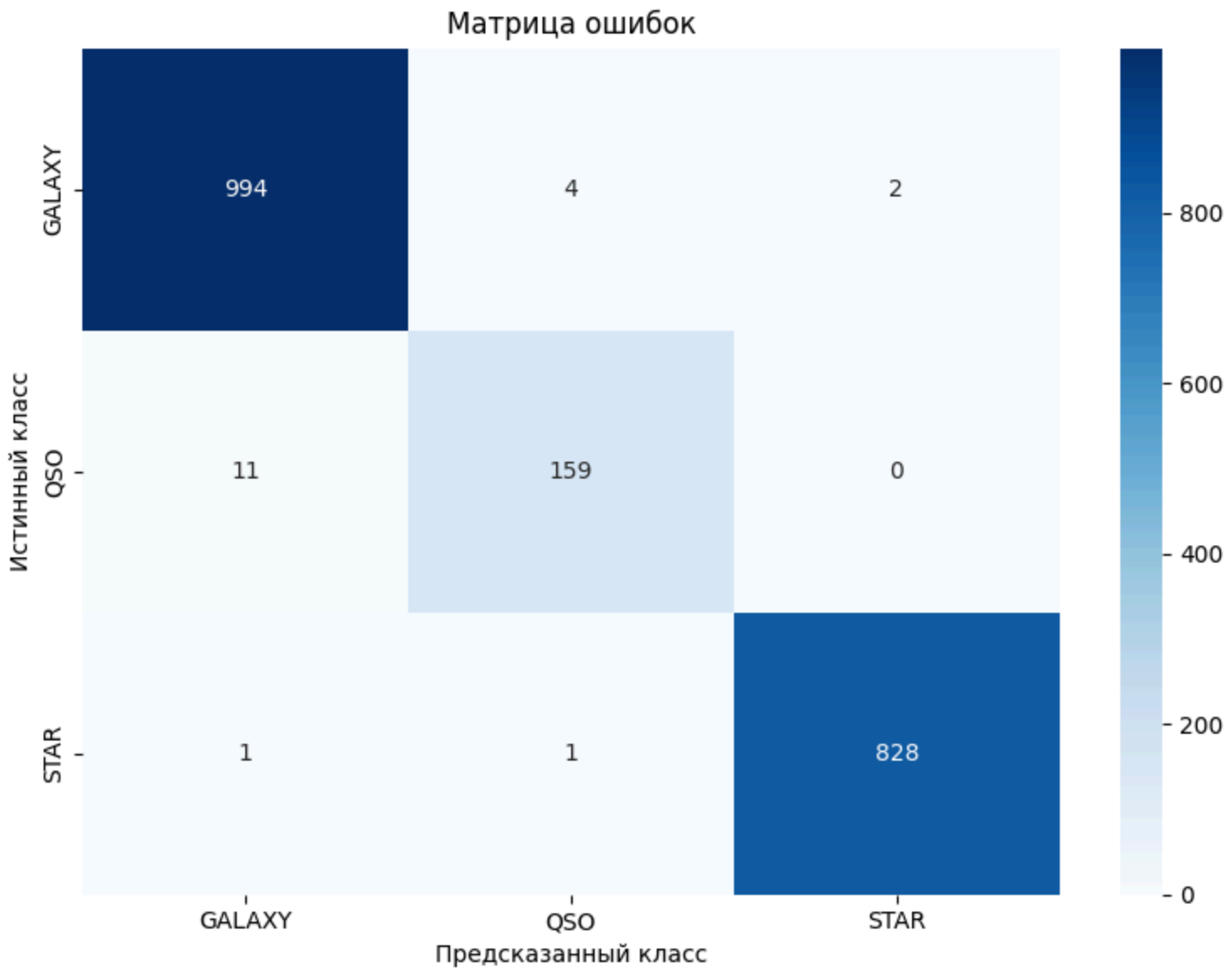
```
Лучшие параметры: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
Лучший score на кросс-валидации: 0.9798

Результаты гипотезы 2:
1. Accuracy: 0.9920

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.989066 0.995000  0.992024
   QSO   0.981595 0.941176  0.960961
  STAR   0.997593 0.998795  0.998194

3. Macro F1: 0.9837

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     995    3     2
QSO         10  160     0
STAR         1    0   829
```



Гипотеза 3: Формирование новых признаков улучшит качество модели.

In [16]:
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_class_h3_poly = poly.fit_transform(X_train_class_h1_scaled)
X_test_class_h3_poly = poly.transform(X_test_class_h1_scaled)

gb_classifier_h3 = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_classifier_h3.fit(X_train_class_h3_poly, y_train_class_h1)
y_pred_class_h3 = gb_classifier_h3.predict(X_test_class_h3_poly)

print("Результаты гипотезы 3:")
class_metrics_h3 = evaluate_classification_model(y_test_class_h1, y_pred_class_h3, class_names)
```

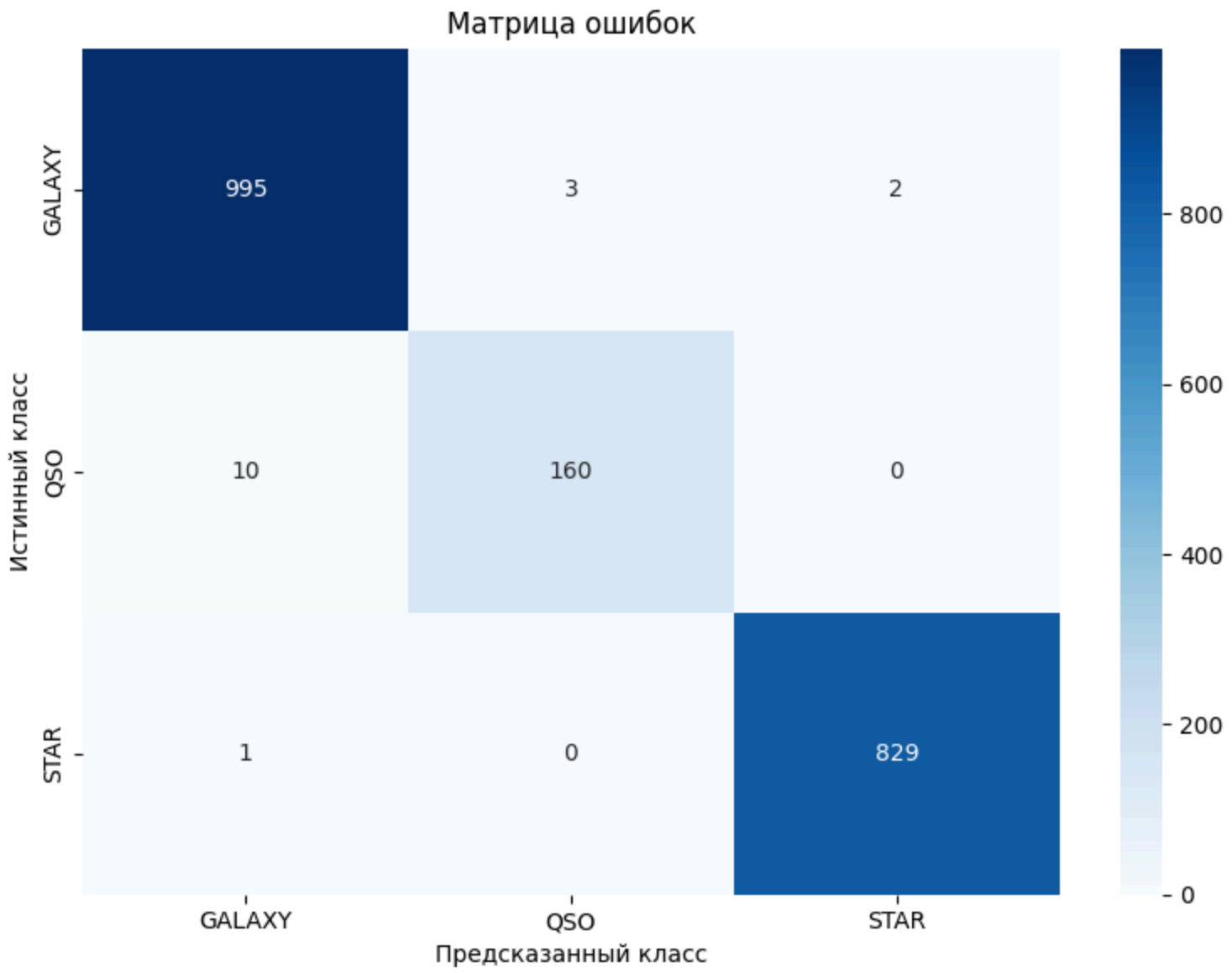
```
Результаты гипотезы 3:
1. Accuracy: 0.9905

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.988072 0.994000  0.991027
   QSO   0.969512 0.935294  0.952096
  STAR   0.997590 0.997590  0.997590

3. Macro F1: 0.9802

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     994    4     2
QSO         11  159     0
STAR         1    1   828
```



Сформируем улучшенный бейзлайн по результатам проверки гипотез. Выберем лучшую комбинацию техник.

In [17]:
```
X_train_class_improved, X_test_class_improved, y_train_class_improved, y_test_class_improved = train_test_s
    X_class, y_class_encoded, test_size=0.2, random_state=42, stratify=y_class_encoded
)

scaler_class_improved = StandardScaler()
X_train_class_improved_scaled = scaler_class_improved.fit_transform(X_train_class_improved)
X_test_class_improved_scaled = scaler_class_improved.transform(X_test_class_improved)

gb_classifier_improved = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
gb_classifier_improved.fit(X_train_class_improved_scaled, y_train_class_improved)
y_pred_class_improved = gb_classifier_improved.predict(X_test_class_improved_scaled)

print("Результаты улучшенного бейзлайна:")
class_metrics_improved = evaluate_classification_model(y_test_class_improved, y_pred_class_improved, class_
```

```
Результаты улучшенного бейзлайна:
1. Accuracy: 0.9920

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.989066 0.995000  0.992024
   QSO   0.981595 0.941176  0.960961
  STAR   0.997593 0.998795  0.998194

3. Macro F1: 0.9837

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     995    3     2
QSO         10  160     0
STAR         1    0   829
```



Сравним результаты моделей с улучшенным бейзлайном в сравнении с результатами

In [18]:
```
comparison_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'Accuracy': [class_metrics['accuracy'], class_metrics_improved['accuracy']],
    'Macro F1': [class_metrics['macro_f1'], class_metrics_improved['macro_f1']]
})

print("Сравнение базового и улучшенного бейзлайна для классификации:")
print(comparison_class.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для классификации:
            Модель  Accuracy  Macro F1
   Базовый бейзлайн    0.9925  0.985590
Улучшенный бейзлайн    0.9920  0.983726
```

Регрессия

Гипотеза 1: Добавление стандартизации признаков улучшит качество модели.

In [19]:
```
scaler_reg_h1 = StandardScaler()
X_train_reg_h1_scaled = scaler_reg_h1.fit_transform(X_train_reg)
X_test_reg_h1_scaled = scaler_reg_h1.transform(X_test_reg)

gb_regressor_h1 = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_regressor_h1.fit(X_train_reg_h1_scaled, y_train_reg)
y_pred_reg_h1 = gb_regressor_h1.predict(X_test_reg_h1_scaled)

print("Результаты гипотезы 1:")
reg_metrics_h1 = evaluate_regression_model(y_test_reg, y_pred_reg_h1)
```

```
Результаты гипотезы 1:
1. MAE: 1.5748

2. MSE: 5.0377

3. RMSE: 2.2445

4. R²: 0.5346
```

Гипотеза 2: Подбор гиперпараметров на кросс-валидации улучшит качество модели.

In [20]:
```
param_grid_reg_h2 = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5]
}

gb_regressor_h2 = GradientBoostingRegressor(random_state=42)
grid_search_reg_h2 = GridSearchCV(gb_regressor_h2, param_grid_reg_h2, cv=5, scoring='r2', n_jobs=-1)
grid_search_reg_h2.fit(X_train_reg_h1_scaled, y_train_reg)

print(f"Лучшие параметры: {grid_search_reg_h2.best_params_}")
```

```
print(f"Лучший score на кросс-валидации: {grid_search_reg_h2.best_score_:.4f}")

y_pred_reg_h2 = grid_search_reg_h2.predict(X_test_reg_h1_scaled)

print("\nРезультаты гипотезы 2:")
reg_metrics_h2 = evaluate_regression_model(y_test_reg, y_pred_reg_h2)
```

```
Лучшие параметры: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Лучший score на кросс-валидации: 0.5509

Результаты гипотезы 2:
1. MAE: 1.5748

2. MSE: 5.0377

3. RMSE: 2.2445

4. R²: 0.5346
```

Гипотеза 3: Формирование новых признаков улучшит качество модели.

In [21]:
```
poly_reg = PolynomialFeatures(degree=2, include_bias=False)
X_train_reg_h3_poly = poly_reg.fit_transform(X_train_reg_h1_scaled)
X_test_reg_h3_poly = poly_reg.transform(X_test_reg_h1_scaled)

gb_regressor_h3 = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_regressor_h3.fit(X_train_reg_h3_poly, y_train_reg)
y_pred_reg_h3 = gb_regressor_h3.predict(X_test_reg_h3_poly)

print("Результаты гипотезы 3:")
reg_metrics_h3 = evaluate_regression_model(y_test_reg, y_pred_reg_h3)
```

```
Результаты гипотезы 3:
1. MAE: 1.5692

2. MSE: 5.0521

3. RMSE: 2.2477

4. R²: 0.5333
```

Сформируем улучшенный бейзлайн по результатам проверки гипотез. Выберем лучшую комбинацию техник.

In [22]:
```
scaler_reg_improved = StandardScaler()
X_train_reg_improved_scaled = scaler_reg_improved.fit_transform(X_train_reg)
X_test_reg_improved_scaled = scaler_reg_improved.transform(X_test_reg)

gb_regressor_improved = GradientBoostingRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
gb_regressor_improved.fit(X_train_reg_improved_scaled, y_train_reg)
y_pred_reg_improved = gb_regressor_improved.predict(X_test_reg_improved_scaled)

print("Результаты улучшенного бейзлайна:")
reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_reg_improved)
```

```
Результаты улучшенного бейзлайна:
1. MAE: 1.5941

2. MSE: 5.2179

3. RMSE: 2.2843

4. R²: 0.5180
```

Сравним результаты моделей с улучшенным бейзлайном в сравнении с результатами

In [23]:
```
comparison_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн', 'Улучшенный бейзлайн'],
    'MAE': [reg_metrics['mae'], reg_metrics_improved['mae']],
    'RMSE': [reg_metrics['rmse'], reg_metrics_improved['rmse']],
    'R²': [reg_metrics['r2'], reg_metrics_improved['r2']]
})

print("Сравнение базового и улучшенного бейзлайна для регрессии:")
print(comparison_reg.to_string(index=False))
```

```
Сравнение базового и улучшенного бейзлайна для регрессии:
            Модель      MAE     RMSE       R²
   Базовый бейзлайн 1.574048 2.244432 0.534653
Улучшенный бейзлайн 1.594142 2.284274 0.517986
```

4. Имплементация алгоритма машинного обучения

Перейдем к имлпементации алгоритмов

Классификация

Реализуем алгоритм градиентного бустинга для классификации

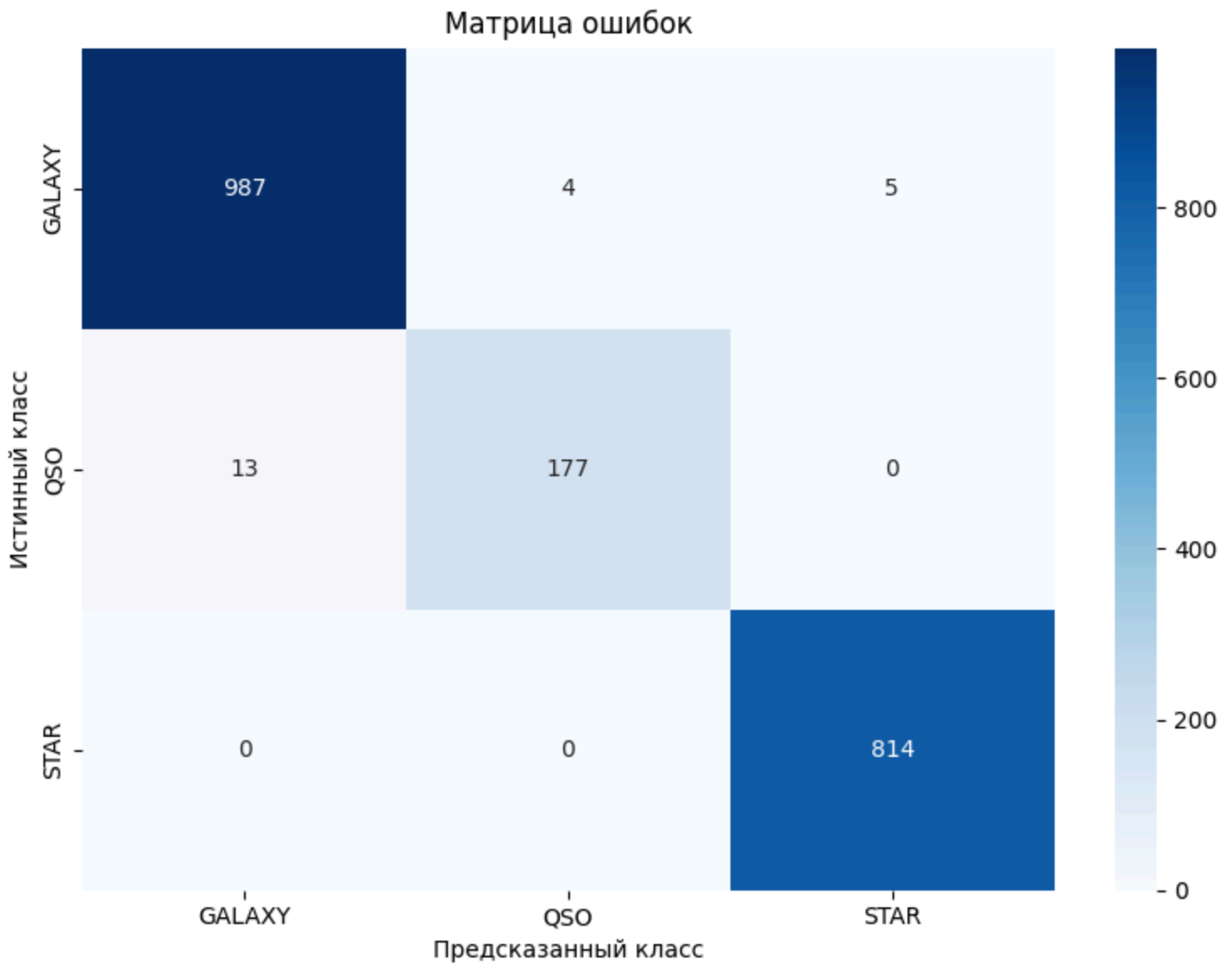
In [24]:
```
from sklearn.tree import DecisionTreeRegressor

class MyGradientBoostingClassifier:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, random_state=None):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.random_state = random_state
        self.estimators_ = []
        self.initial_prediction_ = None
        self.classes_ = None
        self.n_classes_ = None

    def _softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / (np.sum(exp_z, axis=1, keepdims=True) + 1e-10)

    def _sigmoid(self, z):
        z = np.clip(z, -500, 500)
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)

        if self.random_state is not None:
            np.random.seed(self.random_state)

        self.classes_ = np.unique(y)
        self.n_classes_ = len(self.classes_)

        n_samples, n_features = X.shape

        if self.n_classes_ == 2:
            pos_class_ratio = np.mean(y)
            self.initial_prediction_ = np.log(pos_class_ratio / (1 - pos_class_ratio + 1e-10))
            predictions = np.full(n_samples, self.initial_prediction_)
        else:
            class_counts = np.bincount(y, minlength=self.n_classes_)
            class_probs = class_counts / len(y)
            self.initial_prediction_ = np.log(class_probs + 1e-10)
            predictions = np.tile(self.initial_prediction_, (n_samples, 1))

        for i in range(self.n_estimators):
            if self.n_classes_ == 2:
                proba = self._sigmoid(predictions)
                residuals = y - proba

                tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=self.random_state)
                tree.fit(X, residuals)
                self.estimators_.append(tree)

                tree_pred = tree.predict(X)
                predictions += self.learning_rate * tree_pred
            else:
                proba = self._softmax(predictions)
                residuals = np.zeros((n_samples, self.n_classes_))

                for k in range(self.n_classes_):
                    y_k = (y == self.classes_[k]).astype(float)
                    residuals[:, k] = y_k - proba[:, k]

                trees = []
                for k in range(self.n_classes_):
                    tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=self.random_state)
                    tree.fit(X, residuals[:, k])
                    trees.append(tree)
                self.estimators_.append(trees)

                for k in range(self.n_classes_):
                    tree_pred = trees[k].predict(X)
                    predictions[:, k] += self.learning_rate * tree_pred

        return self

    def predict_proba(self, X):
        X = np.array(X)
        n_samples = X.shape[0]

        if self.n_classes_ == 2:
            predictions = np.full(n_samples, self.initial_prediction_)

            for tree in self.estimators_:
                tree_pred = tree.predict(X)
```

```
                predictions += self.learning_rate * tree_pred

            proba = self._sigmoid(predictions)
            return np.column_stack([1 - proba, proba])
        else:
            predictions = np.tile(self.initial_prediction_, (n_samples, 1))

            for trees in self.estimators_:
                for k, tree in enumerate(trees):
                    tree_pred = tree.predict(X)
                    predictions[:, k] += self.learning_rate * tree_pred

            proba = self._softmax(predictions)
            return proba

    def predict(self, X):
        proba = self.predict_proba(X)

        if self.n_classes_ == 2:
            return (proba[:, 1] > 0.5).astype(int)
        else:
            return self.classes_[np.argmax(proba, axis=1)]
```

Обучим имплементированную модель на исходных данных

In [25]:
```
my_gb_classifier = MyGradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_s
my_gb_classifier.fit(X_train_class.values, y_train_class)
y_pred_my_class = my_gb_classifier.predict(X_test_class.values)

print("Результаты имплементированной модели классификации:")
my_class_metrics = evaluate_classification_model(y_test_class, y_pred_my_class, class_names)
```

```
Результаты имплементированной модели классификации:
1. Accuracy: 0.9890

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.987000 0.990964  0.988978
   QSO   0.977901 0.931579  0.954178
  STAR   0.993895 1.000000  0.996938

3. Macro F1: 0.9800

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     987    4     5
QSO         13  177     0
STAR         0    0   814
```



Сравним результаты имплементированной модели с базовым бейзлайном

In [26]:
```
comparison_my_class = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель базового бейзлайна'],
    'Accuracy': [class_metrics['accuracy'], my_class_metrics['accuracy']],
```

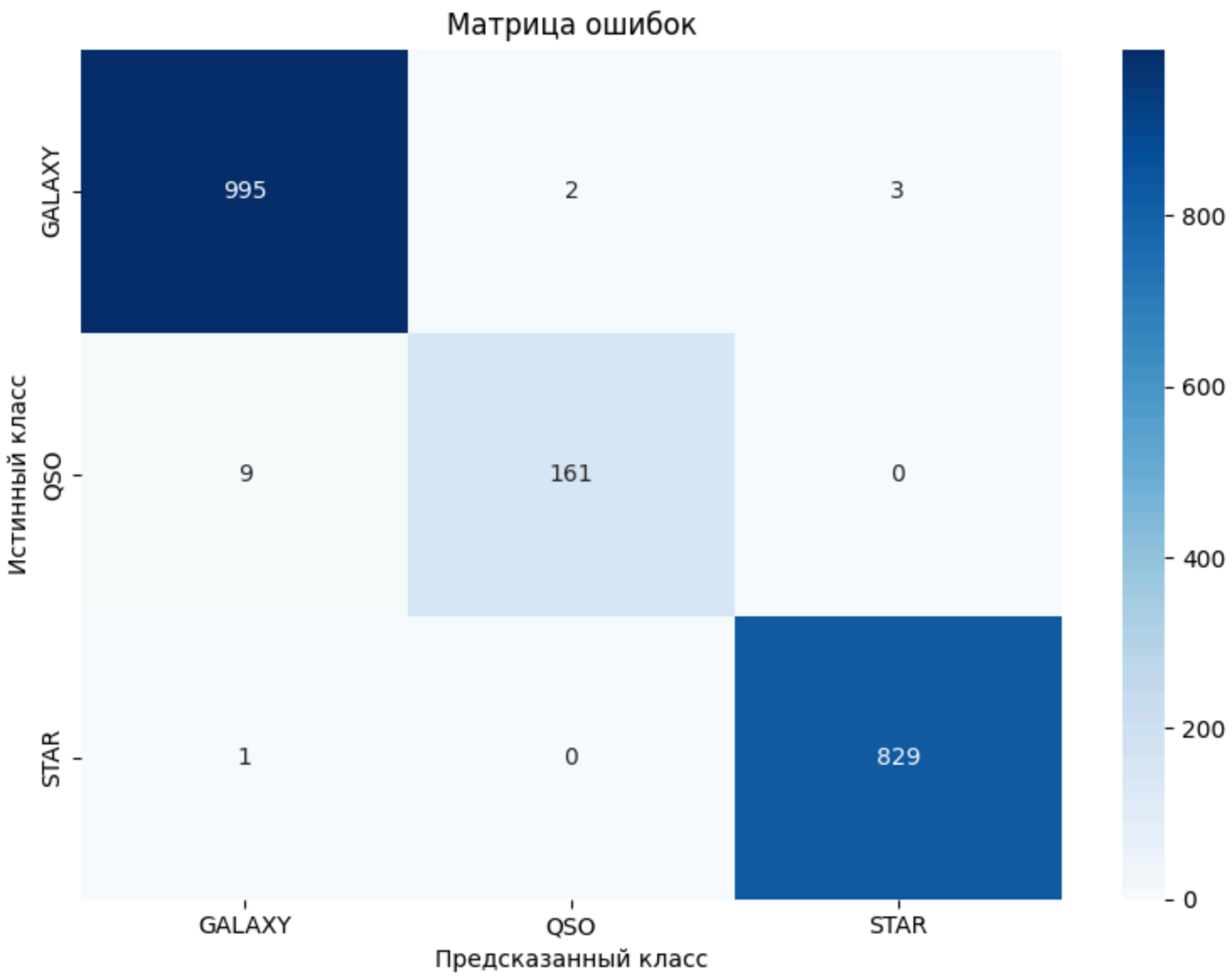
```
    'Macro F1': [class_metrics['macro_f1'], my_class_metrics['macro_f1']]
})

print("Сравнение имплементированной модели с базовым бейзлайном для классификации:")
print(comparison_my_class.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном для классификации:
                                  Модель  Accuracy  Macro F1
                  Базовый бейзлайн (sklearn)    0.9925  0.985590
Имплементированная модель базового бейзлайна    0.9890  0.980031
```

Добавим техники из улучшенного бейзлайна и обучим модель

In [27]:
```
my_gb_classifier_improved = MyGradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
my_gb_classifier_improved.fit(X_train_class_improved_scaled, y_train_class_improved)
y_pred_my_class_improved = my_gb_classifier_improved.predict(X_test_class_improved_scaled)

print("Результаты имплементированной модели классификации (улучшенный вариант):")
my_class_metrics_improved = evaluate_classification_model(y_test_class_improved, y_pred_my_class_improved,
```

```
Результаты имплементированной модели классификации (улучшенный вариант):
1. Accuracy: 0.9925

2. Метрики по классам:
 Класс  Precision   Recall  F1-score
GALAXY   0.990050 0.995000  0.992519
   QSO   0.987730 0.947059  0.966967
  STAR   0.996394 0.998795  0.997593

3. Macro F1: 0.9857

4. Матрица ошибок:
        GALAXY  QSO  STAR
GALAXY     995    2     3
QSO          9  161     0
STAR         1    0   829
```



Сравним результаты моделей в сравнении с результатами

In [28]:
```
comparison_my_class_final = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'Accuracy': [class_metrics_improved['accuracy'], my_class_metrics_improved['accuracy']],
    'Macro F1': [class_metrics_improved['macro_f1'], my_class_metrics_improved['macro_f1']]
})

print("Сравнение имплементированной модели с улучшенным бейзлайном:")
print(comparison_my_class_final.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшенным бейзлайном:
                                Модель  Accuracy  Macro F1
          Улучшенный бейзлайн (sklearn)    0.9920  0.983726
Имплементированная модель с улучшениями    0.9925  0.985693
```

Регрессия

Реализуем алгоритм градиентного бустинга для регрессии

In [29]:
```python
from sklearn.tree import DecisionTreeRegressor

class MyGradientBoostingRegressor:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, random_state=None):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.random_state = random_state
        self.estimators_ = []
        self.initial_prediction_ = None

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)

        if self.random_state is not None:
            np.random.seed(self.random_state)

        self.initial_prediction_ = np.mean(y)
        predictions = np.full(len(y), self.initial_prediction_)

        for i in range(self.n_estimators):
            residuals = y - predictions

            tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=self.random_state)
            tree.fit(X, residuals)
            self.estimators_.append(tree)

            tree_pred = tree.predict(X)
            predictions += self.learning_rate * tree_pred

        return self

    def predict(self, X):
        X = np.array(X)
        predictions = np.full(X.shape[0], self.initial_prediction_)

        for tree in self.estimators_:
            tree_pred = tree.predict(X)
            predictions += self.learning_rate * tree_pred

        return predictions
```

Обучим имплементированную модель на исходных данных

In [30]:
```python
my_gb_regressor = MyGradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_sta
my_gb_regressor.fit(X_train_reg.values, y_train_reg.values)
y_pred_my_reg = my_gb_regressor.predict(X_test_reg.values)

print("Результаты имплементированной модели регрессии:")
my_reg_metrics = evaluate_regression_model(y_test_reg, y_pred_my_reg)
```

```
Результаты имплементированной модели регрессии:
1. MAE: 1.5723

2. MSE: 5.0234

3. RMSE: 2.2413

4. R²: 0.5360
```

Сравним результаты имплементированной модели с базовым бейзлайном

In [31]:
```python
comparison_my_reg = pd.DataFrame({
    'Модель': ['Базовый бейзлайн (sklearn)', 'Имплементированная модель базового бейзлайна'],
    'MAE': [reg_metrics['mae'], my_reg_metrics['mae']],
    'RMSE': [reg_metrics['rmse'], my_reg_metrics['rmse']],
    'R²': [reg_metrics['r2'], my_reg_metrics['r2']]
})

print("Сравнение имплементированной модели с базовым бейзлайном для регрессии:")
print(comparison_my_reg.to_string(index=False))
```

```
Сравнение имплементированной модели с базовым бейзлайном для регрессии:
                                Модель      MAE     RMSE       R²
                 Базовый бейзлайн (sklearn) 1.574048 2.244432 0.534653
Имплементированная модель базового бейзлайна 1.572250 2.241285 0.535957
```

Добавим техники из улучшенного бейзлайна и обучим модель

In [32]:
```
my_gb_regressor_improved = MyGradientBoostingRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
my_gb_regressor_improved.fit(X_train_reg_improved_scaled, y_train_reg.values)
y_pred_my_reg_improved = my_gb_regressor_improved.predict(X_test_reg_improved_scaled)

print("Результаты имплементированной модели регрессии (улучшенный вариант):")
my_reg_metrics_improved = evaluate_regression_model(y_test_reg, y_pred_my_reg_improved)
```

```
Результаты имплементированной модели регрессии (улучшенный вариант):
1. MAE: 1.5964

2. MSE: 5.2455

3. RMSE: 2.2903

4. R²: 0.5154
```

Сравним результаты моделей с результатами

In [33]:
```
comparison_my_reg_final = pd.DataFrame({
    'Модель': ['Улучшенный бейзлайн (sklearn)', 'Имплементированная модель с улучшениями'],
    'MAE': [reg_metrics_improved['mae'], my_reg_metrics_improved['mae']],
    'RMSE': [reg_metrics_improved['rmse'], my_reg_metrics_improved['rmse']],
    'R²': [reg_metrics_improved['r2'], my_reg_metrics_improved['r2']]
})

print("Сравнение имплементированной модели с улучшенным бейзлайном:")
print(comparison_my_reg_final.to_string(index=False))
```

```
Сравнение имплементированной модели с улучшенным бейзлайном:
                              Модель      MAE     RMSE       R²
          Улучшенный бейзлайн (sklearn) 1.594142 2.284274 0.517986
Имплементированная модель с улучшениями 1.596399 2.290297 0.515440
```

Общие выводы по результатам всех моделей

Сравним все 4 модели для классификации и регрессии: базовый бейзлайн из sklearn, имплементированную модель базового бейзлайна, модель с улучшенным бейзлайном из sklearn и имплементированную модель улучшенного бейзлайна.

Классификация

In [34]:
```
final_comparison_class = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'Accuracy': [
        class_metrics['accuracy'],
        my_class_metrics['accuracy'],
        class_metrics_improved['accuracy'],
        my_class_metrics_improved['accuracy']
    ],
    'Macro F1': [
        class_metrics['macro_f1'],
        my_class_metrics['macro_f1'],
        class_metrics_improved['macro_f1'],
        my_class_metrics_improved['macro_f1']
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ")
print(final_comparison_class.to_string(index=False))

print("\n\nДетальное сравнение метрик по классам:")

print("\n1. Базовый бейзлайн (sklearn):")
print(f"   Precision: {class_metrics['precision']}")
print(f"   Recall: {class_metrics['recall']}")
print(f"   F1-score: {class_metrics['f1']}")

print("\n2. Имплементированная модель базового бейзлайна:")
print(f"   Precision: {my_class_metrics['precision']}")
print(f"   Recall: {my_class_metrics['recall']}")
print(f"   F1-score: {my_class_metrics['f1']}")

print("\n3. Улучшенный бейзлайн (sklearn):")
print(f"   Precision: {class_metrics_improved['precision']}")
print(f"   Recall: {class_metrics_improved['recall']}")
print(f"   F1-score: {class_metrics_improved['f1']}")
```

```
print("\n4. Имплементированная модель улучшенного бейзлайна:")
print(f"   Precision: {my_class_metrics_improved['precision']}")
print(f"   Recall: {my_class_metrics_improved['recall']}")
print(f"   F1-score: {my_class_metrics_improved['f1']}")

print("ВЫВОДЫ ПО КЛАССИФИКАЦИИ:")
print(f"1. Базовый бейзлайн (sklearn): Accuracy = {class_metrics['accuracy']:.4f}, Macro F1 = {class_metri
print(f"2. Имплементированная модель базового бейзлайна: Accuracy = {my_class_metrics['accuracy']:.4f}, Ma
print(f"3. Улучшенный бейзлайн (sklearn): Accuracy = {class_metrics_improved['accuracy']:.4f}, Macro F1 = 
print(f"4. Имплементированная модель улучшенного бейзлайна: Accuracy = {my_class_metrics_improved['accurac

print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  Accuracy: {((class_metrics_improved['accuracy'] - class_metrics['accuracy']) / class_metrics['ac
print(f"  Macro F1: {((class_metrics_improved['macro_f1'] - class_metrics['macro_f1']) / class_metrics['ma

print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в Accuracy: {abs(class_metrics_improved['accuracy'] - my_class_metrics_improved['accurac
print(f"  Разница в Macro F1: {abs(class_metrics_improved['macro_f1'] - my_class_metrics_improved['macro_f
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ КЛАССИФИКАЦИИ
                                    Модель  Accuracy  Macro F1
                  Базовый бейзлайн (sklearn)    0.9925  0.985590
   Имплементированная модель базового бейзлайна    0.9890  0.980031
                 Улучшенный бейзлайн (sklearn)    0.9920  0.983726
Имплементированная модель улучшенного бейзлайна    0.9925  0.985693


Детальное сравнение метрик по классам:

1. Базовый бейзлайн (sklearn):
   Precision: [0.99099099 0.97326203 0.9987715 ]
   Recall: [0.9939759  0.95789474 0.9987715 ]
   F1-score: [0.9924812  0.96551724 0.9987715 ]

2. Имплементированная модель базового бейзлайна:
   Precision: [0.987      0.97790055 0.99389499]
   Recall: [0.99096386 0.93157895 1.        ]
   F1-score: [0.98897796 0.9541779  0.99693815]

3. Улучшенный бейзлайн (sklearn):
   Precision: [0.98906561 0.98159509 0.99759326]
   Recall: [0.995      0.94117647 0.99879518]
   F1-score: [0.99202393 0.96096096 0.99819386]

4. Имплементированная модель улучшенного бейзлайна:
   Precision: [0.99004975 0.98773006 0.99639423]
   Recall: [0.995      0.94705882 0.99879518]
   F1-score: [0.9925187  0.96696697 0.99759326]
ВЫВОДЫ ПО КЛАССИФИКАЦИИ:
1. Базовый бейзлайн (sklearn): Accuracy = 0.9925, Macro F1 = 0.9856
2. Имплементированная модель базового бейзлайна: Accuracy = 0.9890, Macro F1 = 0.9800
3. Улучшенный бейзлайн (sklearn): Accuracy = 0.9920, Macro F1 = 0.9837
4. Имплементированная модель улучшенного бейзлайна: Accuracy = 0.9925, Macro F1 = 0.9857

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  Accuracy: -0.05%
  Macro F1: -0.19%

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в Accuracy: 0.000500
  Разница в Macro F1: 0.001967
```

Регрессия

In [35]:

```
final_comparison_reg = pd.DataFrame({
    'Модель': [
        'Базовый бейзлайн (sklearn)',
        'Имплементированная модель базового бейзлайна',
        'Улучшенный бейзлайн (sklearn)',
        'Имплементированная модель улучшенного бейзлайна'
    ],
    'MAE': [
        reg_metrics['mae'],
        my_reg_metrics['mae'],
        reg_metrics_improved['mae'],
        my_reg_metrics_improved['mae']
    ],
    'RMSE': [
        reg_metrics['rmse'],
        my_reg_metrics['rmse'],
        reg_metrics_improved['rmse'],
        my_reg_metrics_improved['rmse']
    ],
    'R²': [
        reg_metrics['r2'],
        my_reg_metrics['r2'],
        reg_metrics_improved['r2'],
        my_reg_metrics_improved['r2']
```

```
    ]
})

print("ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ")
print(final_comparison_reg.to_string(index=False))

print("\nВЫВОДЫ ПО РЕГРЕССИИ:")
print(f"1. Базовый бейзлайн (sklearn): MAE = {reg_metrics['mae']:.4f}, RMSE = {reg_metrics['rmse']:.4f}, R
print(f"2. Имплементированная модель базового бейзлайна: MAE = {my_reg_metrics['mae']:.4f}, RMSE = {my_reg_
print(f"3. Улучшенный бейзлайн (sklearn): MAE = {reg_metrics_improved['mae']:.4f}, RMSE = {reg_metrics_imp
print(f"4. Имплементированная модель улучшенного бейзлайна: MAE = {my_reg_metrics_improved['mae']:.4f}, RM

print("\nУлучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):")
print(f"  MAE: {((reg_metrics_improved['mae'] - reg_metrics['mae']) / reg_metrics['mae'] * 100):.2f}% улучш
print(f"  RMSE: {((reg_metrics_improved['rmse'] - reg_metrics['rmse']) / reg_metrics['rmse'] * 100):.2f}% у
print(f"  R²: {((reg_metrics_improved['r2'] - reg_metrics['r2']) / reg_metrics['r2'] * 100):.2f}% улучшени

print("\nСравнение имплементированной модели с sklearn (улучшенный бейзлайн):")
print(f"  Разница в MAE: {abs(reg_metrics_improved['mae'] - my_reg_metrics_improved['mae']):.6f}")
print(f"  Разница в RMSE: {abs(reg_metrics_improved['rmse'] - my_reg_metrics_improved['rmse']):.6f}")
print(f"  Разница в R²: {abs(reg_metrics_improved['r2'] - my_reg_metrics_improved['r2']):.6f}")
```

```
ОБЩЕЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ РЕГРЕССИИ
                                    Модель      MAE     RMSE       R²
                  Базовый бейзлайн (sklearn) 1.574048 2.244432 0.534653
 Имплементированная модель базового бейзлайна 1.572250 2.241285 0.535957
                 Улучшенный бейзлайн (sklearn) 1.594142 2.284274 0.517986
Имплементированная модель улучшенного бейзлайна 1.596399 2.290297 0.515440

ВЫВОДЫ ПО РЕГРЕССИИ:
1. Базовый бейзлайн (sklearn): MAE = 1.5740, RMSE = 2.2444, R² = 0.5347
2. Имплементированная модель базового бейзлайна: MAE = 1.5723, RMSE = 2.2413, R² = 0.5360
3. Улучшенный бейзлайн (sklearn): MAE = 1.5941, RMSE = 2.2843, R² = 0.5180
4. Имплементированная модель улучшенного бейзлайна: MAE = 1.5964, RMSE = 2.2903, R² = 0.5154

Улучшение базового бейзлайна (sklearn) → улучшенный бейзлайн (sklearn):
  MAE: 1.28% улучшение
  RMSE: 1.78% улучшение
  R²: -3.12% улучшение

Сравнение имплементированной модели с sklearn (улучшенный бейзлайн):
  Разница в MAE: 0.002256
  Разница в RMSE: 0.006023
  Разница в R²: 0.002545
```

Сравнение результатов, полученных всеми алгоритмами из лабораторных 1-5

In [36]:

```
print("Классификация")

classification_all = {
    'Алгоритм': [
        'KNN - Базовый (Лаб. 1)',
        'KNN - Имплементация (Лаб. 1)',
        'KNN - Улучшенный (Лаб. 1)',
        'KNN - Имплементация улучш. (Лаб. 1)',

        'Логист. регр. - Базовый (Лаб. 2)',
        'Логист. регр. - Имплементация (Лаб. 2)',
        'Логист. регр. - Улучшенный (Лаб. 2)',
        'Логист. регр. - Имплементация улучш. (Лаб. 2)',

        'Дерево - Базовый (Лаб. 3)',
        'Дерево - Имплементация (Лаб. 3)',
        'Дерево - Улучшенный (Лаб. 3)',
        'Дерево - Имплементация улучш. (Лаб. 3)',

        'Лес - Базовый (Лаб. 4)',
        'Лес - Имплементация (Лаб. 4)',
        'Лес - Улучшенный (Лаб. 4)',
        'Лес - Имплементация улучш. (Лаб. 4)',

        'Бустинг - Базовый (Лаб. 5)',
        'Бустинг - Имплементация (Лаб. 5)',
        'Бустинг - Улучшенный (Лаб. 5)',
        'Бустинг - Имплементация улучш. (Лаб. 5)',
    ],
    'Accuracy': [
        0.9485, 0.9485, 0.9615, 0.9615,
        0.9575, 0.4980, 0.9890, 0.7985,
        0.9885, 0.9875, 0.9905, 0.9915,
        0.9900, 0.8935, 0.9915, 0.9400,
        0.9925, 0.9890, 0.9920, 0.9925,
    ],
    'Macro F1': [
        0.953601, 0.953601, 0.963740, 0.963740,
        0.955983, 0.221629, 0.985502, 0.826120,
        0.978467, 0.975035, 0.981024, 0.981758,
```

```
        0.983610, 0.899846, 0.986153, 0.935364,
        0.984941, 0.980031, 0.983726, 0.985693,
    ]
}

df_class_all = pd.DataFrame(classification_all)

print(df_class_all.to_string(index=False))

baseline_names = ['KNN', 'Логист. регр.', 'Дерево', 'Лес', 'Бустинг']
baseline_acc = [0.9485, 0.9575, 0.9885, 0.9900, 0.9925]
impl_acc = [0.9485, 0.4980, 0.9875, 0.8935, 0.9890]
improved_acc = [0.9615, 0.9890, 0.9905, 0.9915, 0.9920]
impl_improved_acc = [0.9615, 0.7985, 0.9915, 0.9400, 0.9925]

print("\n\nРезультаты по типам реализации\n")
print("Базовые бейзлайны sklearn:")
for name, acc in zip(baseline_names, baseline_acc):
    print(f"  {name:<20} {acc:.4f}")

print("\nИмплементированные модели базового бейзлайна:")
for name, acc in zip(baseline_names, impl_acc):
    print(f"  {name:<20} {acc:.4f}")

print("\nУлучшенные бейзлайны sklearn:")
for name, acc in zip(baseline_names, improved_acc):
    print(f"  {name:<20} {acc:.4f}")

print("\nИмплементированные улучшенные модели:")
for name, acc in zip(baseline_names, impl_improved_acc):
    print(f"  {name:<20} {acc:.4f} ")

print("\n\n Статистика по типам реализации")
print(f"  Базовые бейзлайны (sklearn)        | Средняя Accuracy: {np.mean(baseline_acc):.4f}, Макс: {np.ma
print(f"  Имплементированные базовые        | Средняя Accuracy: {np.mean(impl_acc):.4f}, Макс: {np.max(imp
print(f"  Улучшенные бейзлайны (sklearn)    | Средняя Accuracy: {np.mean(improved_acc):.4f}, Макс: {np.max
print(f"  Имплементированные улучшенные    | Средняя Accuracy: {np.mean(impl_improved_acc):.4f}, Макс: {np

print("\n\n Лучшие результаты:")
best_overall_idx = df_class_all['Accuracy'].idxmax()
print(f"  Лучший результат (Accuracy): {df_class_all.loc[best_overall_idx, 'Алгоритм']}")
print(f"    Accuracy: {df_class_all.loc[best_overall_idx, 'Accuracy']:.4f}")
print(f"    Macro F1: {df_class_all.loc[best_overall_idx, 'Macro F1']:.4f}")

print("\n\n Лучший результат для каждого алгоритма")
algos = ['KNN', 'Логист. регр.', 'Дерево', 'Лес', 'Бустинг']
indices = [(0, 3), (4, 7), (8, 11), (12, 15), (16, 19)]
for algo, (start, end) in zip(algos, indices):
    max_idx = df_class_all.loc[start:end, 'Accuracy'].idxmax()
    model_name = df_class_all.loc[max_idx, 'Алгоритм'].split(' - ')[-1]
    acc = df_class_all.loc[max_idx, 'Accuracy']
    print(f"  {algo:<20} {model_name:<30} Accuracy: {acc:.4f}")

print("Регрессия")

regression_all = {
    'Алгоритм': [
        'KNN - Базовый (Лаб. 1)',
        'KNN - Имплементация (Лаб. 1)',
        'KNN - Улучшенный (Лаб. 1)',
        'KNN - Имплементация улучш. (Лаб. 1)',

        'Линейная регр. - Базовый (Лаб. 2)',
        'Линейная регр. - Имплементация (Лаб. 2)',
        'Линейная регр. - Улучшенный (Лаб. 2)',
        'Линейная регр. - Имплементация улучш. (Лаб. 2)',

        'Дерево - Базовый (Лаб. 3)',
        'Дерево - Имплементация (Лаб. 3)',
        'Дерево - Улучшенный (Лаб. 3)',
        'Дерево - Имплементация улучш. (Лаб. 3)',

        'Лес - Базовый (Лаб. 4)',
        'Лес - Имплементация (Лаб. 4)',
        'Лес - Улучшенный (Лаб. 4)',
        'Лес - Имплементация улучш. (Лаб. 4)',

        'Бустинг - Базовый (Лаб. 5)',
        'Бустинг - Имплементация (Лаб. 5)',
        'Бустинг - Улучшенный (Лаб. 5)',
        'Бустинг - Имплементация улучш. (Лаб. 5)',
    ],
    'MAE': [
        1.555981, 1.555981, 1.575222, 1.575222,
        1.630561, 1.943738, 1.630633, 1.675446,
        2.178230, 2.216507, 1.624829, 1.625966,
        1.589928, 1.782342, 1.519789, 1.609885,
        1.574048, 1.572250, 1.594142, 1.596399,
    ],
    'RMSE': [
```

```
        2.265278, 2.265278, 2.263895, 2.263895,
        2.250008, 2.691109, 2.250000, 2.327788,
        3.133589, 3.157167, 2.317374, 2.317720,
        2.256425, 2.501466, 2.153528, 2.277700,
        2.244432, 2.241285, 2.284274, 2.290297,
    ],
    'R²': [
        0.525969, 0.525969, 0.526548, 0.526548,
        0.532338, 0.331000, 0.532342, 0.499447,
        0.092916, 0.079214, 0.503915, 0.503767,
        0.529667, 0.421967, 0.571585, 0.520756,
        0.534653, 0.535957, 0.517986, 0.515440,
    ]
}

df_reg_all = pd.DataFrame(regression_all)

print(df_reg_all.to_string(index=False))

baseline_mae = [1.555981, 1.630561, 2.178230, 1.589928, 1.574048]
impl_mae = [1.555981, 1.943738, 2.216507, 1.782342, 1.572250]
improved_mae = [1.575222, 1.630633, 1.624829, 1.519789, 1.594142]
impl_improved_mae = [1.575222, 1.675446, 1.625966, 1.609885, 1.596399]

baseline_r2 = [0.525969, 0.532338, 0.092916, 0.529667, 0.534653]
impl_r2 = [0.525969, 0.331000, 0.079214, 0.421967, 0.535957]
improved_r2 = [0.526548, 0.532342, 0.503915, 0.571585, 0.517986]
impl_improved_r2 = [0.526548, 0.499447, 0.503767, 0.520756, 0.515440]
```

```
print("\n\nРезультаты по типам реализации\n")
print("Базовые бейзлайны sklearn:")
for name, r2 in zip(baseline_names, baseline_r2):
    print(f"  {name:<20} R²: {r2:.4f}")

print("\nИмплементированные модели базового бейзлайна:")
for name, r2 in zip(baseline_names, impl_r2):
    print(f"  {name:<20} R²: {r2:.4f}")

print("\nУлучшенные бейзлайны sklearn:")
for name, r2 in zip(baseline_names, improved_r2):
    print(f"  {name:<20} R²: {r2:.4f}")

print("\nИмплементированные улучшенные модели:")
for name, r2 in zip(baseline_names, impl_improved_r2):
    print(f"  {name:<20} R²: {r2:.4f}")

print("\n\nСтатистика по типам реализации:")
print(f"  Базовые бейзлайны (sklearn)        | Средняя R²: {np.mean(baseline_r2):.4f}, Макс: {np.max(baseli
print(f"  Имплементированные базовые         | Средняя R²: {np.mean(impl_r2):.4f}, Макс: {np.max(impl_r2):.4
print(f"  Улучшенные бейзлайны (sklearn)     | Средняя R²: {np.mean(improved_r2):.4f}, Макс: {np.max(improve
print(f"  Имплементированные улучшенные      | Средняя R²: {np.mean(impl_improved_r2):.4f}, Макс: {np.max(imp

print("\n\n Лучшие результаты R²:")
best_reg_idx = df_reg_all['R²'].idxmax()
print(f"  Лучший результат: {df_reg_all.loc[best_reg_idx, 'Алгоритм']}")
print(f"    R²: {df_reg_all.loc[best_reg_idx, 'R²']:.4f}")
print(f"    MAE: {df_reg_all.loc[best_reg_idx, 'MAE']:.4f}")
print(f"    RMSE: {df_reg_all.loc[best_reg_idx, 'RMSE']:.4f}")

print("\n\n Лучший результат для каждого алгоритма:")
for algo, (start, end) in zip(algos, indices):
    max_idx = df_reg_all.loc[start:end, 'R²'].idxmax()
    model_name = df_reg_all.loc[max_idx, 'Алгоритм'].split(' - ')[-1]
    r2 = df_reg_all.loc[max_idx, 'R²']
    mae = df_reg_all.loc[max_idx, 'MAE']
    print(f"  {algo:<20} {model_name:<30} R²: {r2:.4f}, MAE: {mae:.4f}")
```

Классификация

Алгоритм	Accuracy	Macro F1
KNN - Базовый (Лаб. 1)	0.9485	0.953601
KNN - Имплементация (Лаб. 1)	0.9485	0.953601
KNN - Улучшенный (Лаб. 1)	0.9615	0.963740
KNN - Имплементация улучш. (Лаб. 1)	0.9615	0.963740
Логист. регр. - Базовый (Лаб. 2)	0.9575	0.955983
Логист. регр. - Имплементация (Лаб. 2)	0.4980	0.221629
Логист. регр. - Улучшенный (Лаб. 2)	0.9890	0.985502
Логист. регр. - Имплементация улучш. (Лаб. 2)	0.7985	0.826120
Дерево - Базовый (Лаб. 3)	0.9885	0.978467
Дерево - Имплементация (Лаб. 3)	0.9875	0.975035
Дерево - Улучшенный (Лаб. 3)	0.9905	0.981024
Дерево - Имплементация улучш. (Лаб. 3)	0.9915	0.981758
Лес - Базовый (Лаб. 4)	0.9900	0.983610
Лес - Имплементация (Лаб. 4)	0.8935	0.899846
Лес - Улучшенный (Лаб. 4)	0.9915	0.986153
Лес - Имплементация улучш. (Лаб. 4)	0.9400	0.935364
Бустинг - Базовый (Лаб. 5)	0.9925	0.984941
Бустинг - Имплементация (Лаб. 5)	0.9890	0.980031
Бустинг - Улучшенный (Лаб. 5)	0.9920	0.983726
Бустинг - Имплементация улучш. (Лаб. 5)	0.9925	0.985693

Результаты по типам реализации

Базовые бейзлайны sklearn:
- KNN 0.9485
- Логист. регр. 0.9575
- Дерево 0.9885
- Лес 0.9900
- Бустинг 0.9925

Имплементированные модели базового бейзлайна:
- KNN 0.9485
- Логист. регр. 0.4980
- Дерево 0.9875
- Лес 0.8935
- Бустинг 0.9890

Улучшенные бейзлайны sklearn:
- KNN 0.9615
- Логист. регр. 0.9890
- Дерево 0.9905
- Лес 0.9915
- Бустинг 0.9920

Имплементированные улучшенные модели:
- KNN 0.9615
- Логист. регр. 0.7985
- Дерево 0.9915
- Лес 0.9400
- Бустинг 0.9925

Статистика по типам реализации
- Базовые бейзлайны (sklearn) | Средняя Accuracy: 0.9754, Макс: 0.9925, Мин: 0.9485
- Имплементированные базовые | Средняя Accuracy: 0.8633, Макс: 0.9890, Мин: 0.4980
- Улучшенные бейзлайны (sklearn) | Средняя Accuracy: 0.9849, Макс: 0.9920, Мин: 0.9615
- Имплементированные улучшенные | Средняя Accuracy: 0.9368, Макс: 0.9925, Мин: 0.7985

Лучшие результаты:
- Лучший результат (Accuracy): Бустинг - Базовый (Лаб. 5)
  - Accuracy: 0.9925
  - Macro F1: 0.9849

Лучший результат для каждого алгоритма

KNN	Улучшенный (Лаб. 1)	Accuracy: 0.9615
Логист. регр.	Улучшенный (Лаб. 2)	Accuracy: 0.9890
Дерево	Имплементация улучш. (Лаб. 3)	Accuracy: 0.9915
Лес	Улучшенный (Лаб. 4)	Accuracy: 0.9915
Бустинг	Базовый (Лаб. 5)	Accuracy: 0.9925

Регрессия

Алгоритм	MAE	RMSE	R^2
KNN - Базовый (Лаб. 1)	1.555981	2.265278	0.525969
KNN - Имплементация (Лаб. 1)	1.555981	2.265278	0.525969
KNN - Улучшенный (Лаб. 1)	1.575222	2.263895	0.526548
KNN - Имплементация улучш. (Лаб. 1)	1.575222	2.263895	0.526548
Линейная регр. - Базовый (Лаб. 2)	1.630561	2.250008	0.532338
Линейная регр. - Имплементация (Лаб. 2)	1.943738	2.691109	0.331000
Линейная регр. - Улучшенный (Лаб. 2)	1.630633	2.250000	0.532342
Линейная регр. - Имплементация улучш. (Лаб. 2)	1.675446	2.327788	0.499447
Дерево - Базовый (Лаб. 3)	2.178230	3.133589	0.092916
Дерево - Имплементация (Лаб. 3)	2.216507	3.157167	0.079214
Дерево - Улучшенный (Лаб. 3)	1.624829	2.317374	0.503915
Дерево - Имплементация улучш. (Лаб. 3)	1.625966	2.317720	0.503767
Лес - Базовый (Лаб. 4)	1.589928	2.256425	0.529667
Лес - Имплементация (Лаб. 4)	1.782342	2.501466	0.421967

```
                     Лес - Улучшенный (Лаб. 4) 1.519789 2.153528 0.571585
            Лес - Имплементация улучш. (Лаб. 4) 1.609885 2.277700 0.520756
                      Бустинг - Базовый (Лаб. 5) 1.574048 2.244432 0.534653
               Бустинг - Имплементация (Лаб. 5) 1.572250 2.241285 0.535957
                  Бустинг - Улучшенный (Лаб. 5) 1.594142 2.284274 0.517986
         Бустинг - Имплементация улучш. (Лаб. 5) 1.596399 2.290297 0.515440


Результаты по типам реализации

Базовые бейзлайны sklearn:
  KNN                  R²: 0.5260
  Логист. регр.        R²: 0.5323
  Дерево               R²: 0.0929
  Лес                  R²: 0.5297
  Бустинг              R²: 0.5347

Имплементированные модели базового бейзлайна:
  KNN                  R²: 0.5260
  Логист. регр.        R²: 0.3310
  Дерево               R²: 0.0792
  Лес                  R²: 0.4220
  Бустинг              R²: 0.5360

Улучшенные бейзлайны sklearn:
  KNN                  R²: 0.5265
  Логист. регр.        R²: 0.5323
  Дерево               R²: 0.5039
  Лес                  R²: 0.5716
  Бустинг              R²: 0.5180

Имплементированные улучшенные модели:
  KNN                  R²: 0.5265
  Логист. регр.        R²: 0.4994
  Дерево               R²: 0.5038
  Лес                  R²: 0.5208
  Бустинг              R²: 0.5154


Статистика по типам реализации:
  Базовые бейзлайны (sklearn)        | Средняя R²: 0.4431, Макс: 0.5347, Мин: 0.0929
  Имплементированные базовые         | Средняя R²: 0.3788, Макс: 0.5360, Мин: 0.0792
  Улучшенные бейзлайны (sklearn)     | Средняя R²: 0.5305, Макс: 0.5716, Мин: 0.5039
  Имплементированные улучшенные      | Средняя R²: 0.5132, Макс: 0.5265, Мин: 0.4994


 Лучшие результаты R²:
  Лучший результат: Лес - Улучшенный (Лаб. 4)
    R²: 0.5716
    MAE: 1.5198
    RMSE: 2.1535


 Лучший результат для каждого алгоритма:
  KNN                  Улучшенный (Лаб. 1)           R²: 0.5265, MAE: 1.5752
  Логист. регр.        Улучшенный (Лаб. 2)           R²: 0.5323, MAE: 1.6306
  Дерево               Улучшенный (Лаб. 3)           R²: 0.5039, MAE: 1.6248
  Лес                  Улучшенный (Лаб. 4)           R²: 0.5716, MAE: 1.5198
  Бустинг              Имплементация (Лаб. 5)        R²: 0.5360, MAE: 1.5722
```