ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		В.В. Боженко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

КЛАССИФИКАЦИЯ 2024

по курсу: ВВЕДЕНИЕ В АНАЛИЗ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4217		Д.М. Никитин
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1. **Цель работы:** изучение алгоритмов и методов классификации на практике.

2. **Вариант и задания:**

Вариант 4.

Данные о болезнях сердца:

1. возраст: возраст пациента (лет)

2. анемия: снижение количества эритроцитов или гемоглобина (логическое значение)

3. высокое кровяное давление: если у пациента гипертония (логическое значение)

4. креатининфосфокиназа (КФК): уровень фермента КФК в крови (мкг/л)

5. диабет: если у пациента диабет (логическое значение)

6. фракция выброса: процент крови, покидающей сердце при каждом сокращении (в процентах)

7. тромбоциты: тромбоциты в крови (килотромбоциты/ мл)

8. пол: женщина или мужчина (бинарный)

9. креатинин сыворотки: уровень креатинина сыворотки в крови (мг/дл)

10. натрий сыворотки: уровень натрия сыворотки в крови (мэкв/л)

11. курение: если пациент курит или нет (логическое)

12. время: период наблюдения (дни)

13. событие смерти: если пациент умер в течение периода наблюдения (логическое значение)

Порядок выполнения

1. Загрузить набор данных.

2. Провести предварительную обработку данных.

3. Выделить целевую переменную, которую необходимо предсказать. Не включать эту целевую

переменную в модель. Построить матрицу диаграмм рассеяния,

выделив значения целевой переменной

разными цветами.

4. Разбить набор данных на тренировочной и тестовый датасеты с помощью train_test_split и выполнить

стандартизацию числовых данных с помощью StandardScaler.

5. Для получения оценки 5 - разработать предсказательную модель качественного отклика методами:

метод k- ближайших соседей

дерево решений

логистическая регрессия

случайный лес.

Для получения оценки 4 - разработать предсказательную модель качественного отклика любыми

двумя методами.

6. Оценить ошибку классификации для каждого метода. Подсчитать метрики "Accuracy", "Precision",

"Recall", "Balanced accuracy", 'F1 score'.

7. Построить матрицу неточностей с помощью confusion_matrix для каждого метода.

8. Построить графики ROC-кривой для каждого метода на одном графике (4 линии на одном графике) для

сравнения.

9. Сделать вывод о качестве построенного классификатора по подсчитанным выше метрикам.

3. **Ход работы:**

Сначала был считан датафрейм 4heart2.csv с помощью pd.readcsv(). И проведена проверка на наличие пропусков и дубликатов явных и не явных. Реализацию и результат выполнения можно увидеть на рисунках 1, 2, 3 и 4.

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, balanced_accuracy_score, f1_score, confusion_matrix, roc_curve, auc

# Считывание файла csv
df = pd.read_csv("4heart2.csv")

# Подсчёт явных дубликатов и пропущенных ячеек
num_duplicates = df.duplicated().sum()
num_missing_values = df.isnull().sum().sum()

print(f"Количество пустых ячеек: {num_missing_values}")
print(f"Количество дубликатов: {num_duplicates}")

# Получения информации для вывода о не явных дубликатах
for column in df.columns:
    print(f"Уникальные значения в колонке: {column}")
    print(df[column].unique())
```
✓ [161] 25ms

Рисунок 1 – Считывание и предварительная обработка данных

```
Количество пустых ячеек: 0
Количество дубликатов: 0
Уникальные значения в колонке: age
[75.    55.    65.    50.    90.    60.    80.    62.    45.    49.
 82.    87.    70.    48.    68.    53.    95.    58.    94.    85.
 69.    72.    51.    57.    42.    41.    67.    79.    59.    44.
 63.    86.    66.    43.    46.    61.    81.    52.    64.    40.
 60.667 73.    77.    78.    54.    47.    56.   ]
Уникальные значения в колонке: anaemia
[0 1]
Уникальные значения в колонке: creatinine_phosphokinase
[ 582 7861  146  111  160   47  246  315  157  123   81  231  981  168
   80  379  149  125   52  128  220   63  148  112  122   60   70   23
  249  159   94  855 2656  235  124  571  127  588 1380  553  129  577
   91 3964   69  260  371   75  607  789  364 7702  318  109   68  250
  110  161  113 5882  224   92  102  203  336   76   55  280   78   84
  115   66  897  154  144  133  514   59  156   61  305  898 5209   53
  328  748 1876  936  292  369  143  754  400   96  737  358  200  248
  270 1808 1082  719  193 4540  646  281 1548  805  291  482  943  185
  132 1610 2261  233   30 1846  335   58  910   72  130 2334 2442  776
  196  835 3966  171  198   95 1419  478  176  395   99  145  104 1896
  151  244   62  121  418  167 1211 1767  308   97   64  101  212 2281
  972  131  135 1202  427 1021  118   86  675   57 2794   56  211  166
   93  707  119  232  720  180   90 1185 2017  624  207 2522  572  245
   88  446  191  326  655  258  298 1199  213  257  618 1051 2695 1688
   54  170  253  892  337  615  320  190  103 1820 2060 2413]
Уникальные значения в колонке: diabetes
[0 1]
```

Рисунок 2 – Вывод вышеописанного кода (часть 1)

```
Уникальные значения в колонке: ejection_fraction
[20 38 40 15 60 65 35 25 30 50 14 55 45 62 80 17 70]
Уникальные значения в колонке: high_blood_pressure
[1 0]
Уникальные значения в колонке: platelets
[265000.   263358.03 162000.   210000.   327000.   204000.   127000.
 454000.   388000.   368000.   253000.   136000.   276000.   427000.
  47000.   262000.   166000.   237000.    87000.   297000.   289000.
 149000.   196000.   284000.   153000.   200000.   360000.   319000.
 302000.   188000.   228000.   226000.   321000.   305000.   329000.
 185000.   218000.   194000.   310000.   271000.   451000.   140000.
 395000.   418000.   351000.   255000.   461000.   223000.   216000.
 254000.   390000.   385000.   119000.   213000.   274000.   244000.
 497000.   374000.   122000.   243000.   266000.   317000.   283000.
 324000.   293000.   172000.   406000.   173000.   304000.   235000.
 181000.   249000.   219000.   318000.   221000.   298000.   286000.
 621000.   263000.   850000.   306000.   252000.   328000.   164000.
 507000.   203000.   217000.   300000.   267000.   227000.   250000.
 295000.   231000.   211000.   348000.   229000.   338000.   242000.
 225000.   184000.   277000.   362000.   174000.   448000.    75000.
 334000.   192000.   220000.    70000.   270000.   325000.   176000.
 189000.   281000.   337000.   105000.   132000.   279000.   303000.
 224000.   389000.   365000.   201000.   275000.   350000.   309000.
 260000.   160000.   126000.   259000.    73000.   377000.   212000.
 186000.   268000.   147000.   481000.   290000.   358000.   151000.
 371000.   130000.   504000.   141000.    62000.   330000.   248000.
 257000.   533000.   264000.   282000.   314000.   246000.   301000.
 404000.   236000.   294000.   233000.   308000.   198000.   208000.
```

Рисунок 3 – Вывод вышеописанного кода (часть 2)

```
Уникальные значения в колонке: serum_creatinine
[1.9  1.1  1.3  2.7  2.1  1.2  1.5  9.4  4.   0.9  1.   0.8  1.6  1.83
 5.8  3.   3.5  2.3  0.6  4.4  1.4  6.8  2.2  2.   1.18 2.9  0.7  1.7
 2.5  1.8  3.2  0.75 3.7  3.4  6.1  2.4  9.   5.   0.5  3.8 ]
Уникальные значения в колонке: serum_sodium
[130 136 129 137 116 132 131 138 133 140 127 121 135 134 144 128 145 142
 139 146 141 143 126 124 113 125 148]
Уникальные значения в колонке: sex
[1 0]
Уникальные значения в колонке: smoking
[0 1]
Уникальные значения в колонке: time
[  4   6   7   8  10  11  12  13  14  15  16  20  22  23  24  26  27  28
  29  30  31  32  33  35  38  40  41  42  43  44  45  50  54  55  59  60
  61  63  64  65  66  67  68  71  72  73  74  75  76  77  78  79  80  82
  83  85  86  87  88  90  91  94  95  96  97 100 104 105 106 107 108 109
 110 111 112 113 115 117 118 119 120 121 123 126 129 130 134 135 140 145
 146 147 148 150 154 162 170 171 172 174 175 180 185 186 187 188 192 193
 194 195 196 197 198 200 201 205 206 207 208 209 210 211 212 213 214 215
 216 220 230 231 233 235 237 240 241 244 245 246 247 250 256 257 258 270
 271 278 280 285]
Уникальные значения в колонке: DEATH_EVENT
[1 0]
```

Рисунок 4 – Вывод вышеописанного кода (часть 3)

По результатам проверки было выяснено, что дубликаты и пропуски отсутствуют. Далее была выделена целевая переменная DEATH_EVENT и

простроена матрица диаграмм рассеяния. Реализация и результат представлены на рисунке 5 и 6.

```
# Выделение целевой переменной
target = df['DEATH_EVENT']

# Выделение не целевых данных
features = df.drop(columns=['DEATH_EVENT'])

# Построение матрицы диаграмм рассеяния
sns.pairplot(df, hue='DEATH_EVENT', palette='Set1', diag_kind='kde')
plt.suptitle('Матрица диаграмм рассеяния с окраской по DEATH_EVENT')
# Сохранение для добавления в отчёт
plt.savefig('scatter_matrix.png')
plt.show()
✓ [162] 58s 664ms
```

Рисунок 5 – Реализация выделения целевой переменной и построения диаграммы рассеяния

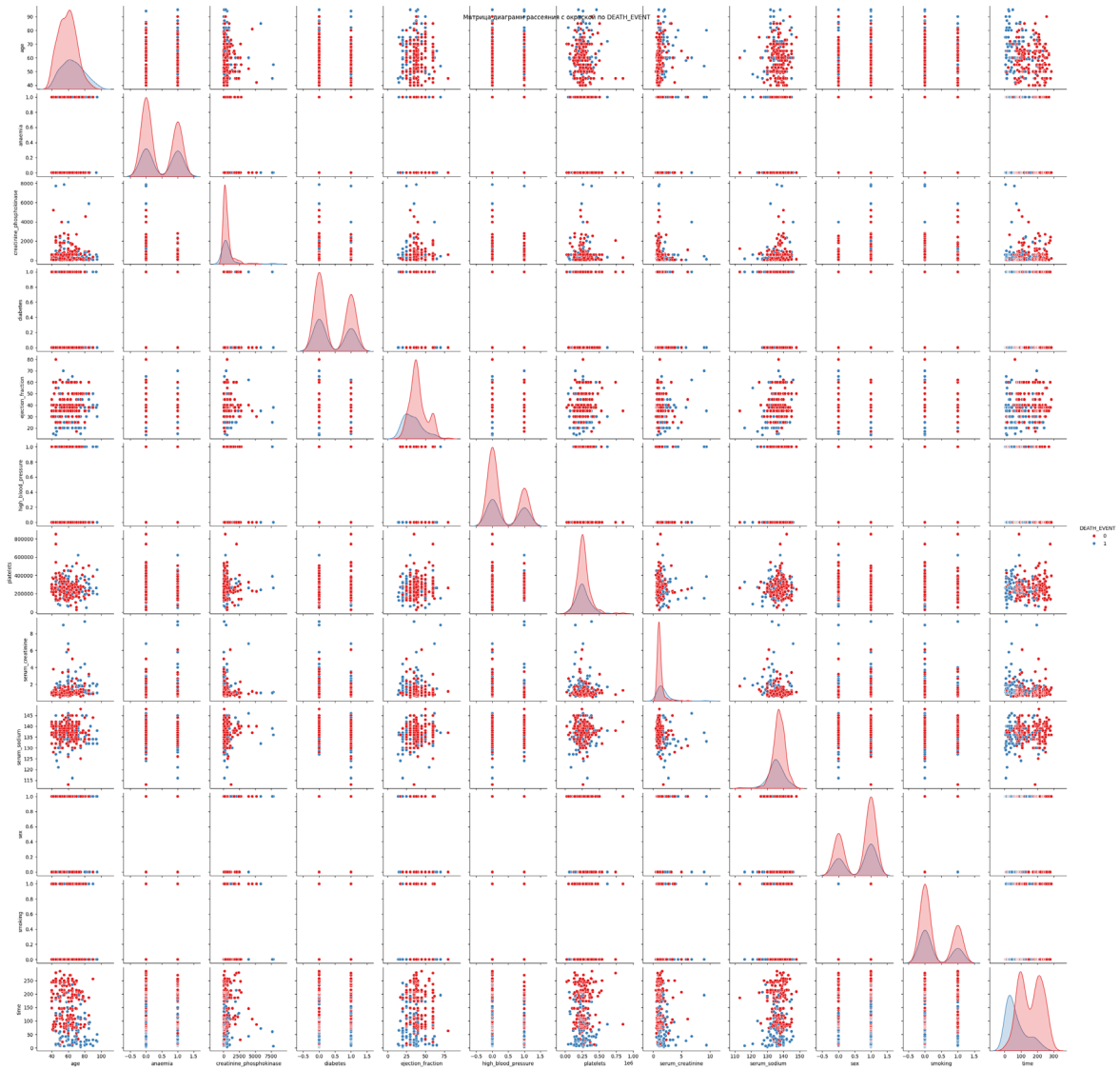

Рисунок 6 – Матрица диаграмм рассеяния

Матрица диаграмм рассеяния показывает, как признаки связаны между собой и как они распределяются относительно целевой переменной DEATH_EVENT. Видно, что классы частично разделимы, но в некоторых случаях сильно пересекаются, что говорит о том, что задача классификации не из простых. Есть признаки, которые могут быть полезными для предсказания, но для более точного анализа придется использовать дополнительные методы, например, корреляцию или построение сложных моделей. В целом, график помогает понять структуру данных и наметить дальнейшие шаги в исследовании.

Далее набор данных был разбит на обучающие и тестовые датасеты. После этого стандартизирован. Реализация этих процессов показана на рисунке 7.

```
# Разбиение на тренировочные и тестовые данные
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.25, random_state=0)

## Стандартизация числовых данных

#Создание стандартного скейлера
scaler = StandardScaler()

# Список числовых признаков для стандартизации
numeric_features = ["age", "creatinine_phosphokinase", "ejection_fraction",
                    "platelets", "serum_creatinine", "serum_sodium", "time"]

# Стандартизация числовых признаков отдельно для тренировочного и тестового набора
X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])
X_test[numeric_features] = scaler.transform(X_test[numeric_features])

# Результаты
# Тренировочные данные: X_train, y_train
# Тестовые данные: X_test, y_test
✓ [163] 20ms
```

Рисунок 7 – Реализация разбиения на группы данных и скейлинга

Данные были разбиты в пропорции 3/1 для тренировочных и тестовых соответственно. Заскейлены были только числовые данные, значения bool не изменялись. Далее были реализованы методы классификаций и получены их метрики. См. рис. 8, 9.

```python
# Метод ближайших соседей
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn_y_pred = knn.predict(X_test)

# Дерево решений
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train, y_train)
dt_y_pred = dt.predict(X_test)

# Логистическая регрессия
lr = LogisticRegression(random_state=0, max_iter=10000)
lr.fit(X_train, y_train)
lr_y_pred = lr.predict(X_test)

# Случайный лес
rf = RandomForestClassifier(random_state=0, n_estimators=100)
rf.fit(X_train, y_train)
rf_y_pred = rf.predict(X_test)

for name, data in {"Метод бижайших соседей": knn_y_pred, "Дерево решений": dt_y_pred,
                   "Логистическая регрессия": lr_y_pred, "Случайный лес": rf_y_pred}.items():
    print(name)
    print(f"accuracy_score: {accuracy_score(y_test, data)}")
    print(f"precision_score: {precision_score(y_test, data)}")
    print(f"recall_score: {recall_score(y_test, data)}")
    print(f"f1_score: {f1_score(y_test, data)}")
    print(f"balanced_accuracy_score: {balanced_accuracy_score(y_test, data)}\n")
✓ [164] 279ms
```

Рисунок 8 – Реализация методов классификации и получения их метрик

```
Метод бижайших соседей
accuracy_score: 0.8
precision_score: 0.8
recall_score: 0.5925925925925926
f1_score: 0.6808510638297872
balanced_accuracy_score: 0.7546296296296295

Дерево решений
accuracy_score: 0.7733333333333333
precision_score: 0.7083333333333334
recall_score: 0.6296296296296297
f1_score: 0.6666666666666666
balanced_accuracy_score: 0.7418981481481481

Логистическая регрессия
accuracy_score: 0.7733333333333333
precision_score: 0.7777777777777778
recall_score: 0.5185185185185185
f1_score: 0.6222222222222222
balanced_accuracy_score: 0.7175925925925926

Случайный лес
accuracy_score: 0.84
precision_score: 0.8947368421052632
recall_score: 0.6296296296296297
f1_score: 0.7391304347826086
```

Рисунок 9 – Полученные метрики

Accuracy (Точность) – это общая доля правильных предсказаний. Высокая точность желательна, но важно помнить, что для несбалансированных классов (например, если один класс встречается гораздо чаще другого) высокая точность не всегда означает хорошую модель.

Precision (Точность предсказания) показывает, насколько точно модель предсказывает положительные классы. Это особенно важно, если ошибка "ложноположительный" (когда модель ошибочно предсказывает положительный класс) имеет серьезные последствия. Например, в медицинской диагностике ложноположительные результаты могут привести к ненужным лечению.

Recall (Полнота) показывает, насколько хорошо модель находит все положительные классы, т.е. долю истинных положительных среди всех положительных наблюдений. Высокий recall важен, если вы хотите не пропустить положительные случаи (например, в медицине, где важно не пропустить болезнь).

F1 Score – это гармоническое среднее между precision и recall. Это полезная метрика, когда важно сбалансировать precision и recall, особенно когда классы несбалансированы. Он помогает избежать ситуации, когда модель имеет высокое значение одной метрики, но низкое значение другой.

Balanced Accuracy (Сбалансированная точность) – это среднее значение точности для каждого класса. Это особенно полезно в случае несбалансированных данных, где класс с малым количеством наблюдений может быть проигнорирован, если используется обычная accuracy.

Метрики

Наибольшее Accuracy имеет случайный лес 0.84.

Наибольшее Precision имеет случайный лес 0.89.

Наибольшее Recall имеет дерево решений и случайный лес 0.63.

Наибольшее F1 Score имеет случайный лес 0.74.

Наибольшее Balanced Accuracy Score имеет случайный лес 0.79.

Ранжирование

Случайный лес — лучшая модель по всем меткам. Метод ближайших соседей — находится на втором месте. Дерево решений — стабильно хорошая модель, но уступает KNN по меткам точности. Логистическая регрессия — хорошая модель, но результаты по полноте поиска положительных классов могли быть и лучше.

Далее были созданы матрицы неточностей. Их реализацию и вывод можно увидеть на рисунке 10 и 11.

```python
cm_knn = confusion_matrix(y_test, knn_y_pred)
cm_dt = confusion_matrix(y_test, dt_y_pred)
cm_lr = confusion_matrix(y_test, lr_y_pred)
cm_rf = confusion_matrix(y_test, rf_y_pred)

confusion_dict = {"Метод бижайших соседей": cm_knn, "Дерево решений": cm_dt,
                  "Логистическая регрессия": cm_lr, "Случайный лес": cm_rf}

print("confusion_matrix")
for name, data in confusion_dict.items():
    print(name)
    print("    0  1")
    print(f"T0->{data[0][0]} {data[0][1]}<-F1")
    print(f"F0->{data[1][0]} {data[1][1]}<-T1")
    print()
```

✓ [165] 12ms

Рисунок 10 – Реализация матриц неточностей

```
confusion_matrix
Метод бижайших соседей
    0  1
T0->44 4<-F1
F0->11 16<-T1

Дерево решений
    0  1
T0->41 7<-F1
F0->10 17<-T1

Логистическая регрессия
    0  1
T0->44 4<-F1
F0->13 14<-T1

Случайный лес
    0  1
T0->46 2<-F1
F0->10 17<-T1
```

Рисунок 11 – Вывод матриц неточностей

Случайный лес выглядит самой точной моделью среди представленных, поскольку имеет наименьшее количество ошибок для класса "0" и достаточно хорошую классификацию для класса "1". Дерево решений и логистическая регрессия дают схожие результаты, но немного хуже, чем случайный лес, особенно для класса "1", где количество ложных отрицаний выше. Метод ближайших соседей имеет наибольшее количество ошибок, особенно для положительного класса, что делает его менее подходящим для данной задачи по сравнению с другими методами, однако списывать его со счетов ещё рано.

Далее были построены графики ROC-кривой для каждого метода на одном графике (4 линии на одном графике) для сравнения. См. рис. 12, 13, 14.

```python
# Предсказания вероятностей положительного класса
knn_y_prob = knn.predict_proba(X_test)[:, 1]
dt_y_prob = dt.predict_proba(X_test)[:, 1]
lr_y_prob = lr.predict_proba(X_test)[:, 1]
rf_y_prob = rf.predict_proba(X_test)[:, 1]

# ROC-кривые и AUC
models = {
    "Случайный лес": rf_y_prob,
    "Метод ближайших соседей": knn_y_prob,
    "Логистическая регрессия": lr_y_prob,
    "Дерево решений": dt_y_prob
}

plt.figure(figsize=(10, 8))

range_master = {}

for name, y_prob in models.items():

    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    label = f'{name} (AUC = {roc_auc:.3f})'
    plt.plot(fpr, tpr, label=label)

# Диагональ случайной классификации
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

```

Рисунок 12 – Реализация кривых ROC и получение коэффициента AUC (часть 1)

```
29  # Оформление графика
30  plt.title('ROC-кривые для методов классификации')
31  plt.xlabel('False Positive Rate')
32  plt.ylabel('True Positive Rate')
33  plt.legend(loc='lower right')
34  plt.grid()
35  plt.savefig('roc_curve.png')
36  plt.show()
    ✓ [166] 494ms
```

Рисунок 13 – Реализация кривых ROC и получение коэффициента AUC (часть 2)

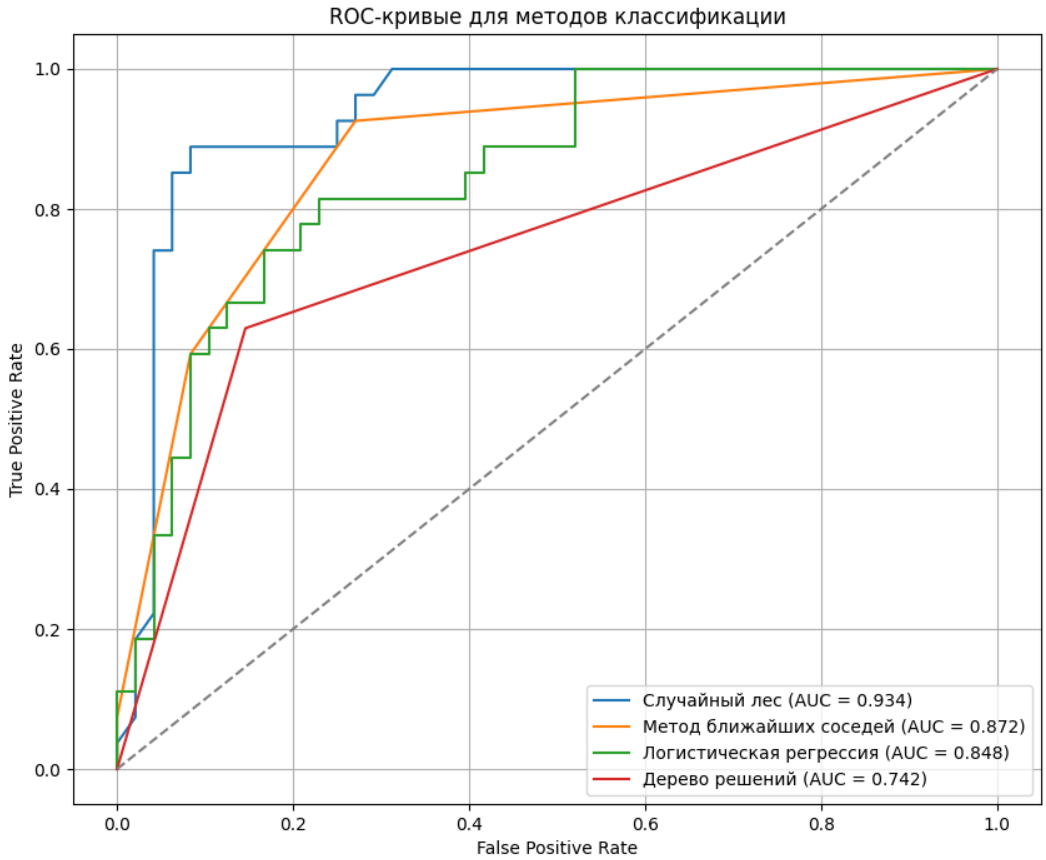

Рисунок 14 – ROC-кривые и коэффициенты AUC

AUC – это метрика, которая оценивает качество классификатора на всех возможных порогах, определяющих границу между положительным и отрицательным классом. AUC измеряет способность модели правильно

классифицировать объекты, при этом значение AUC варьируется от 0 до 1. Чем выше значение AUC, тем лучше модель различает положительные и отрицательные классы.

Случайный лес имеет наилучший показатель AUC (0.934), что означает, что он наиболее точно классифицирует как положительные, так и отрицательные классы. Метод ближайших соседей (KNN) и логистическая регрессия показывают хорошие результаты (AUC 0.872 и 0.848 соответственно), но всё же не достигают уровня случайного леса. Дерево решений с AUC 0.742 имеет наихудший показатель среди рассмотренных моделей, что говорит о том, что оно менее эффективно в решении задачи классификации на данном наборе данных.

4. **Ссылка на Google Colab:**

https://colab.research.google.com/drive/1Gt-eKpDPO66EBJvItsb1LFnLEmJPjnIG?usp=sharing

5. **Расширенный вывод:**

На основании подсчитанных метрик можно сделать следующие выводы о качестве классификаторов:

Случайный лес лучший по всем метрикам: высокая точность (accuracy), сбалансированная метрика (balanced accuracy), а также высокие значения AUC (0.934) и F1-score. Модель хорошо справляется как с классификацией положительных, так и отрицательных примеров. Рекомендуется для использования, если важна высокая точность предсказания и есть достаточно ресурсов для работы с этим более сложным методом.

Метод ближайших соседей занимает уверенное второе место: AUC = 0.872, F1-score и другие метрики указывают на хорошее качество. Немного уступает случайному лесу в способности балансировать между ложноположительными и ложноотрицательными ошибками. Подходит для задач, где важна простота метода и интерпретация.

Логистическая регрессия имеет AUC = 0.848, что указывает на хорошую способность различать классы. F1-score немного ниже, что

указывает на возможную проблему с балансом классов. Рекомендуется, если важна интерпретация модели и простота её работы.

Дерево решений имеет самое слабое качество классификации: AUC = 0.742, F1-score и точность также ниже остальных методов. Склонно к переобучению или недообучению при выбранных параметрах. Может быть улучшено с помощью ансамблевых методов, таких как случайный лес или бустинг.

В итоге: для более точных вычислений рекомендуется использовать случайный лес, для более быстрых (но точных) вычислений рекомендуется использовать логистическую регрессию для малых наборов данных, либо метод ближайших соседей для больших наборов.