



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 Прикладная информатика.

## ОТЧЕТ

по лабораторной работе № 2

Название:

Дисциплина: Прикладной анализ данных

Студент

ИУ6-55Б

(Группа)

\_\_\_\_\_  
(Подпись, дата)

Л.Э. Барсегян

(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

М.А. Кулаев

(И.О. Фамилия)

Москва, 2023

## Вариант 8

**Задание 1. Разделение данных на обучающую (85%) и тестовую часть (15%) случайным образом (можно сделать более корректным методом – разделить в такой пропорции с сохранением распределения таргета в каждой подвыборке).**

В соответствии с вариантом, в лабораторной работе используются данные по следующим районам: Северный, Центральный, Волго-Вятский, Северо-Кавказский, Восточно-Сибирский, Дальневосточный.

При помощи библиотеки Pandas считаем из файла данных целевые признаки и целевые значения и запишем в переменные X и Y соответственно.

Метод `train_test_split`, принимающий X и Y, позволяет разделить данные на обучающую и тестовую выборки. Аргумент `train_size = 0.85` обеспечит 85% обучающей выборки, `random_state` при одном и том же заданном числе воспроизводит одинаковые результаты, `stratify = Y` равномерно распределяет данные между обучающей и тестовой выборками в соответствии с классами Y.

```
# для инициализации случайных чисел
RANDOM_SEED = 46
```

```
df = pd.read_excel('data.xlsx')
X = df.iloc[:, 2:10]
Y = df['Ybin']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.85,
                                                    random_state = RANDOM_SEED,
                                                    stratify = Y)

print('Количество 0 и 1 в Y_train и Y_test:\n')
print(pd.DataFrame([np.bincount(Y_train), np.bincount(Y_test)],
                    columns=['Class 0', 'Class 1'],
                    index=['Y_train', 'Y_test'])))
```

Количество 0 и 1 в Y\_train и Y\_test:

	Class 0	Class 1
Y_train	27	12
Y_test	6	2

Рисунок 1 — разделение данных на выборки

```
print('Количество 0 и 1 в Y_train и Y_test:\n')
print(pd.DataFrame([np.bincount(Y_train), np.bincount(Y_test)],
                    columns=['Class 0', 'Class 1'],
                    index=['Y_train', 'Y_test']))
```

Количество 0 и 1 в Y\_train и Y\_test:

	Class 0	Class 1
Y_train	27	12
Y_test	6	2

Рисунок 2 — проверка стратификации данных в выборках

Как видно на рисунке 2, отношение класса 0 к классу 1 в обучающей выборке равно 2.25, а в тестовой – 3. Разница не сильно большая, можно распределение считать достаточно равномерным.

**Задание 2. Нормирование (масштабирование) исходных данных.**  
**Обратите внимание, что данные для нормализации (масштабирования) рассчитываются только на основе обучающей выборки.**

Прежде всего, необходимо нормировать данные. Для нормализации можно использовать метод “Стандартизация” или “Z-нормализация”, который подходит тем, что централизует данные и менее чувствителен к выбросам, в отличие от метода "Min-Max Scaler".

Для корректной нормализации `std()` и `mean()` берем на основе обучающей выборки.

```
[72] X_mean = X_train.mean()
     X_std = X_train.std()

#нормализация тренировочных и тестовых данных
for col in X_train.columns:
    X_train[col] = (X_train[col] - X_mean[col]) / X_std[col]
    X_test[col] = (X_test[col] - X_mean[col]) / X_std[col]
```

Рисунок 3 — нормализация тренировочных и тестовых данных

**Задание 3. С помощью библиотеки sklearn сделать fit-predict модели kNN. Перебрать по сетке параметр числа соседей с целью определения наилучшего на тестовой выборке.**

Для выполнения задания воспользуемся инструментом GridSearchCV библиотеки scikit-learn, который находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров. Важно отметить, что такой подход может быть весьма времязатратным.

```
# модель
knn = KNeighborsClassifier()
# определение сетки параметров для перебора
grid_params = { 'n_neighbors': list(range(1, 20)) }
# создание объекта GridSearchCV
grid = GridSearchCV(knn, grid_params, cv = 5,
                    scoring = 'accuracy', verbose = 2)

# обучение модели с перебором параметров
knn_model = grid.fit(X_train, Y_train)
```

Рисунок 4 — перебор по сетке параметра числа соседей

На рисунке 4 используется fit-predict модели KNN. Рассмотрим параметры, передающиеся в GridSearchCV:

- KNeighborsClassifier() – метод обучения;
- neighbors: [1, 2, ..., 20] – число соседей для перебора;
- scoring = accuracy – наилучший параметр выбирается по точности;
- cv = 5 – часто используемое значение для кросс-валидации;
- verbose = 2 – для вывода подробной информации о процессе обучения модели (см. рис. 5).

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END .....n_neighbors=1; total time= 0.0s
[CV] END .....n_neighbors=1; total time= 0.0s
[CV] END .....n_neighbors=1; total time= 0.0s
[CV] END .....n_neighbors=1; total time= 0.0s
[CV] END .....n_neighbors=1; total time= 0.0s
[CV] END .....n_neighbors=2; total time= 0.0s
[CV] END .....n_neighbors=2; total time= 0.0s
[CV] END .....n_neighbors=2; total time= 0.0s
[CV] END .....n_neighbors=2; total time= 0.0s
[CV] END .....n_neighbors=2; total time= 0.0s
[CV] END .....n_neighbors=3; total time= 0.0s
[CV] END .....n_neighbors=3; total time= 0.0s
[CV] END .....n_neighbors=3; total time= 0.0s
[CV] END .....n_neighbors=3; total time= 0.0s
[CV] END .....n_neighbors=4; total time= 0.0s
[CV] END .....n_neighbors=4; total time= 0.0s
[CV] END .....n_neighbors=4; total time= 0.0s
[CV] END .....n_neighbors=4; total time= 0.0s
[CV] END .....n_neighbors=5; total time= 0.0s
[CV] END .....n_neighbors=5; total time= 0.0s
[CV] END .....n_neighbors=5; total time= 0.0s
[CV] END .....n_neighbors=5; total time= 0.0s
[CV] END .....n_neighbors=5; total time= 0.0s
[CV] END .....n_neighbors=6; total time= 0.0s
[CV] END .....n_neighbors=6; total time= 0.0s
[CV] END .....n_neighbors=6; total time= 0.0s
[CV] END .....n_neighbors=6; total time= 0.0s
[CV] END .....n_neighbors=6; total time= 0.0s
[CV] END .....n_neighbors=7; total time= 0.0s
```

Рисунок 5 — процесс обучения модели

Как можно заметить на рисунке 5, GridSearchCV для каждого числа соседей переобучается столько раз, сколько было задано в параметре cv. Если модель хорошо работает на обучающем наборе, но плохо на тестовом, это может быть признаком переобучения. Кросс-валидация позволяет выявить такие проблемы и принять меры по их устранению.

```
[6] # Вывод наилучших параметров и оценки
print("Наилучший параметр (число соседей):", knn_model.best_params_)
print("Лучшая оценка (точность (без тестовых данных)):\n
      {:.2f}%".format(knn_model.best_score_ * 100))

# Тестирование модели с наилучшими параметрами
best_knn = knn_model.best_estimator_
Y_predicted = best_knn.predict(X_test)

print(classification_report(Y_test, Y_predicted))
```

Наилучший параметр (число соседей): {'n\_neighbors': 7}  
 Лучшая оценка (точность (без тестовых данных)): 77.50%

	precision	recall	f1-score	support
0	0.83	0.83	0.83	6
1	0.50	0.50	0.50	2
accuracy			0.75	8
macro avg	0.67	0.67	0.67	8
weighted avg	0.75	0.75	0.75	8

Рисунок 6 — вывод результатов обученной модели

Модель показала наибольшую точность в 77,5% при числе соседей  $k = 7$ . Стоит отметить, что точность 77,5% была получена в сетке на основе обучающих данных.

Рассмотрим отчет бинарной классификации подробнее.

- **precision (точность)**: измеряет точность положительных прогнозов модели.

$$precision = \frac{TP}{TP + FP}$$

В нашем случае, получается, для класса 0 83% положительных предсказаний были верными, а для класса 1 – 50%.

- **recall (полнота)**: измеряет способность модели выявить все соответствующие случаи.

$$recall = \frac{TP}{TP + FN}$$

- **F1-Score (F1-мера)**: гармоническое среднее между точностью и полнотой. Она балансирует компромисс между точностью и

полнотой. В вашем случае для класса 0 F1-мера равна 0.83, а для класса 1 - 0.50.

- **support (поддержка)**: представляет собой количество образцов в каждом классе. Для класса 0 есть 6 образцов, а для класса 1 - 2 образца.
- **accuracy (точность)**: общая точность модели, отношение правильно предсказанных случаев ко всем случаям. В вашем случае точность составляет 0.75, что означает, что 75% предсказаний верны.

**Задание 4. С помощью библиотеки sklearn сделать fit-predict модели логистической регрессии. Перебрать по сетке параметр регуляризации с целью определения наилучшего на тестовой выборке.**

Сперва нужно определиться с методом решения логистической регрессии (solver). В документации sklearn метод 'liblinear' рекомендуется для небольших наборов данных, что подходит в нашем случае.

Для каждого метода решения разный набор доступных методов регуляризации.

- 'lbfgs' - ['l2', None]
- 'liblinear' - ['l1', 'l2']
- 'newton-cg' - ['l2', None]
- 'newton-cholesky' - ['l2', None]
- 'sag' - ['l2', None]
- 'saga' - ['elasticnet', 'l1', 'l2', None]

Рисунок 7 — методы регуляризации методов решения

Для метода решения 'liblinear' доступны L1 и L2 регуляризации, поэтому будем перебирать в сетке их.

```

# метод
logistic_regression = LogisticRegression()
# параметры лог. регрессии
grid_params={'penalty': ['l2', 'l1'], 'solver': ['liblinear']}
# создание объекта GridSearchCV
grid = GridSearchCV(logistic_regression, grid_params, cv = 5,
                    scoring = 'accuracy', verbose = 2)

# обучение модели с перебором параметров
logistic_model = grid.fit(X_train, Y_train)

Fitting 5 folds for each of 2 candidates, totalling 10 fits
[CV] END .....penalty=l2, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l2, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l2, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l2, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l2, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l1, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l1, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l1, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l1, solver=liblinear; total time= 0.0s
[CV] END .....penalty=l1, solver=liblinear; total time= 0.0s

```

Рисунок 8 — перебор по сетке параметра регуляризации

```

# Вывод наилучших параметров и оценки
print("Наилучший параметр:", logistic_model.best_params_)
print("Лучшая оценка (точность (без тестовых данных)):\n
      {:.2f}%".format(logistic_model.best_score_ * 100))

# Тестирование модели с наилучшими параметрами
best_logistic = logistic_model.best_estimator_
Y_predicted = logistic_model.predict(X_test)

print(classification_report(Y_test, Y_predicted))

```

```

Наилучший параметр: {'penalty': 'l2'}
Лучшая оценка (точность (без тестовых данных)): 72.14%

```

	precision	recall	f1-score	support
0	0.67	0.33	0.44	6
1	0.20	0.50	0.29	2
accuracy			0.38	8
macro avg	0.43	0.42	0.37	8
weighted avg	0.55	0.38	0.40	8

Рисунок 9 — результаты логистической регрессии



В результате, для  $\lambda_2$  регуляризация дала точность в 72,14% на тренировочных данных, но при этом плохо справилась с тестовыми данными. Такое явление называется переобучением.

С целью избежания переобучения, я попробовала изменить RANDOM\_SEED. Например, с random\_seed = 51 результаты оказались совершенно иными:

```

Наилучший параметр: {'penalty': 'l1', 'solver': 'liblinear'}
Лучшая оценка (точность (без тестовых данных)): 72.14%
      precision    recall  f1-score   support

      0       0.86      1.00      0.92         6
      1       1.00      0.50      0.67         2

 accuracy          0.88         8
 macro avg          0.79         8
weighted avg          0.86         8

```

Рисунок 10 — результаты логистической регрессии  
при другом random\_state

В данном случае наилучшим параметром оказалась L1 регуляризация, поэтому выберем ее. Кроме того, с методом решения liblinear рекомендуют использовать именно L1-регуляризацию.

Замечание. Из-за изменения random\_seed уменьшилась точность в других методах. Было принято решение оставить разные инициализаторы случайных чисел для разных методов.

**Задание 5. С помощью библиотеки sklearn сделать fit-predict модели дерева решений. Перебрать по сетке параметр глубины дерева с целью определения наилучшего на тестовой выборке.**


В модель дерева решений передается параметр splitter = best. Этот вариант заставляет классификатор дерева решений выбирать разделение, которое обеспечивает наилучший прирост информации или уменьшение неопределенности.

```

# метод
decision_tree = DecisionTreeClassifier(splitter = 'best')
# параметры
grid_params={'max_depth': range (1, 100)}
# создание объекта GridSearchCV
grid = GridSearchCV(decision_tree, grid_params, cv = 5,
                    scoring = 'accuracy', verbose = 2)

# обучение модели с перебором параметров
decision_tree_model = grid.fit(X_train, Y_train)

```

 Fitting 5 folds for each of 99 candidates, totalling 495 fits

[CV]	END	.....max_depth=1; total time=	0.0s
[CV]	END	.....max_depth=1; total time=	0.0s
[CV]	END	.....max_depth=1; total time=	0.0s
[CV]	END	.....max_depth=1; total time=	0.0s
[CV]	END	.....max_depth=1; total time=	0.0s
[CV]	END	.....max_depth=2; total time=	0.0s
[CV]	END	.....max_depth=2; total time=	0.0s
[CV]	END	.....max_depth=2; total time=	0.0s
[CV]	END	.....max_depth=2; total time=	0.0s
[CV]	END	.....max_depth=2; total time=	0.0s
[CV]	END	.....max_depth=3; total time=	0.0s
[CV]	END	.....max_depth=3; total time=	0.0s

Рисунок 11 – перебор по сетке модели дерева решений

```

# Вывод наилучших параметров и оценки
print("Наилучший параметр:", decision_tree_model.best_params_)
print("Лучшая оценка (точность (без тестовых данных)):\n{:.2f}%".format(decision_tree_model.best_score_ * 100))

# Тестирование модели с наилучшими параметрами
best_desicion_tree = decision_tree_model.best_estimator_
Y_predicted = best_desicion_tree.predict(X_test)

print(classification_report(Y_test, Y_predicted))

```

Наилучший параметр: {'max\_depth': 3}  
 Лучшая оценка (точность (без тестовых данных)): 69.64%

	precision	recall	f1-score	support
0	0.83	0.83	0.83	6
1	0.50	0.50	0.50	2
accuracy			0.75	8
macro avg	0.67	0.67	0.67	8
weighted avg	0.75	0.75	0.75	8

Рисунок 12 — результаты классификации дерева решений.

Наилучшим параметром оказалась глубина дерева, равная трем.

Аналогично, обучим модель случайного леса.

```
# метод
random_forest = RandomForestClassifier()
# параметры
grid_params={'max_depth': range (1, 21)}
# создание объекта GridSearchCV
grid = GridSearchCV(random_forest, grid_params, cv = 5,
                    scoring = 'accuracy', verbose = 2)

# обучение модели с перебором параметров
random_forest_model = grid.fit(X_train, Y_train)
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

```
[CV] END .....max_depth=1; total time= 0.4s
[CV] END .....max_depth=1; total time= 0.7s
[CV] END .....max_depth=1; total time= 0.5s
[CV] END .....max_depth=1; total time= 0.7s
[CV] END .....max_depth=1; total time= 0.6s
[CV] END .....max_depth=2; total time= 0.6s
[CV] END .....max_depth=2; total time= 1.2s
[CV] END .....max_depth=2; total time= 0.7s
[CV] END .....max_depth=2; total time= 0.7s
[CV] END .....max_depth=2; total time= 0.5s
[CV] END .....max_depth=3; total time= 0.4s
```

Рисунок 13 — перебор по сетке случайно  
леса с параметром максимальной глубины

```
# Вывод наилучших параметров и оценки
print("Наилучший параметр:", random_forest_model.best_params_)
print("Лучшая оценка (точность (без тестовых данных)):\n{:.2f}%".format(random_forest_model.best_score_ * 100))

# Тестирование модели с наилучшими параметрами
best_forest = random_forest_model.best_estimator_
Y_predicted = best_forest.predict(X_test)

print(classification_report(Y_test, Y_predicted))
```

Наилучший параметр: {'max\_depth': 10}  
 Лучшая оценка (точность (без тестовых данных)): 74.64%

	precision	recall	f1-score	support
0	0.75	0.50	0.60	6
1	0.25	0.50	0.33	2
accuracy			0.50	8
macro avg	0.50	0.50	0.47	8
weighted avg	0.62	0.50	0.53	8

Рисунок 14 — результаты модели случайного леса.

При максимальной глубине 10, модель дает точность 50%. Однако, как было замечено ранее, и наилучший параметр, и точность могут измениться в зависимости от разделения исходных данных на выборки.

**Задания 6 - 7. Сравнить качество всех моделей на обучающей и тестовой выборке отдельно по метрикам Accuracy, ROC-AUC, Precision, Recall, F1-мера. Обратите внимание, что 4 из 5 метрик требуют определения порога отсечения по вероятности. В качестве эвристики предлагается взять его как среднее значение полученных вероятностей (желательно, но не обязательно: подобрать по сетке такой порог, при котором precision и recall примерно уравниваются). Проанализировать различие в качестве между моделями. Определить на основе метрик модели, в которых сильно выражено переобучение.**

Посчитаем порог отсечения (threshold) для каждой из модели. Для этого будем для каждого порога считать F1-меру, которая “балансирует” показатели precision и recall. Чем выше значение F1, тем сбалансированнее precision и recall между собой.

При выполнении задания использовался код ниже для каждой модели. В коде изменялась лишь модель.

```
# массив пороговых значений от 0 до 1 с 1000 точками
thresholds = np.linspace(0, 1, 1000)
# список для хранения значений F1-меры
f1_scores = []
# вероятности принадлежности к классу 1 (positive) из модели k-ближайших соседей
y_scores = knn_model.predict_proba(X_test)[: , 1]

# перебираем пороговые значения и вычисляем F1-меру для каждого
for threshold in thresholds:
    y_pred = [1 if score >= threshold else 0 for score in y_scores]
    f1 = f1_score(Y_test, y_pred)
    f1_scores.append(f1)

plt.figure(figsize=(10, 6))
plt.plot(thresholds, f1_scores)
plt.xlabel('Порог')
plt.ylabel('F1-мера')
plt.grid(True)
plt.title('F1-мера по отношению к порогу')
plt.xticks(np.arange(0, 1.1, 0.05))
plt.show()
```

Рисунок 15 — построение графика F1-меры по отношению к порогу отсечения

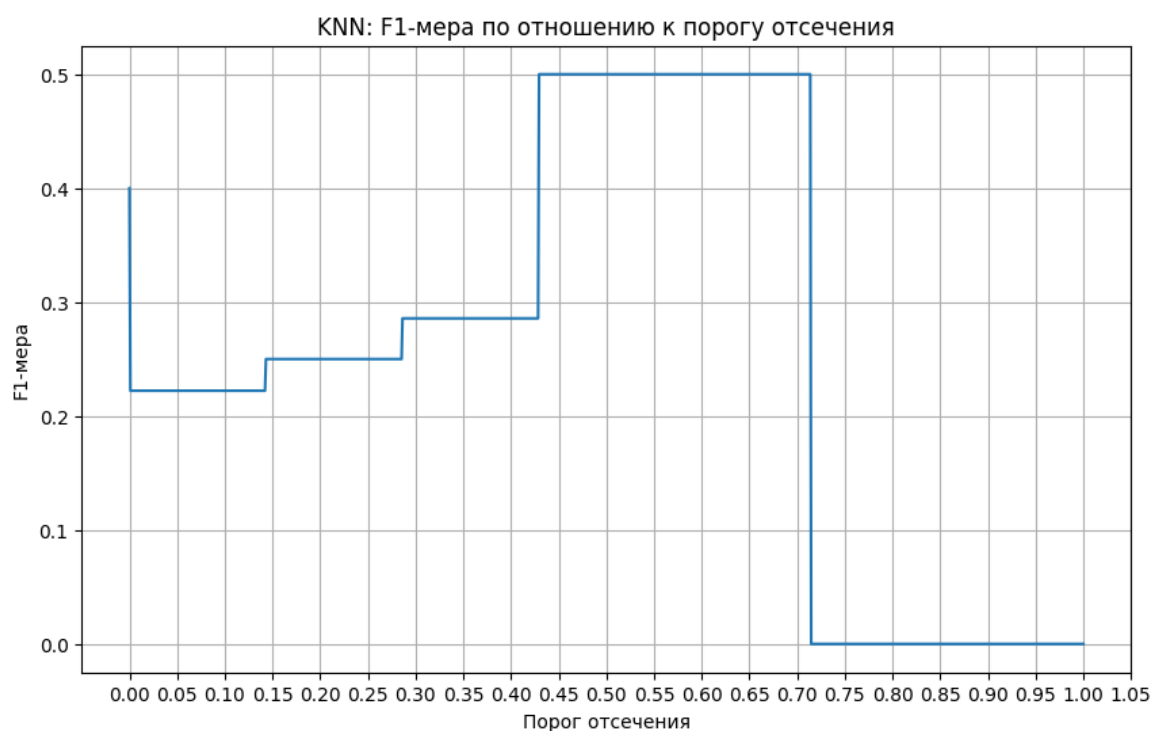
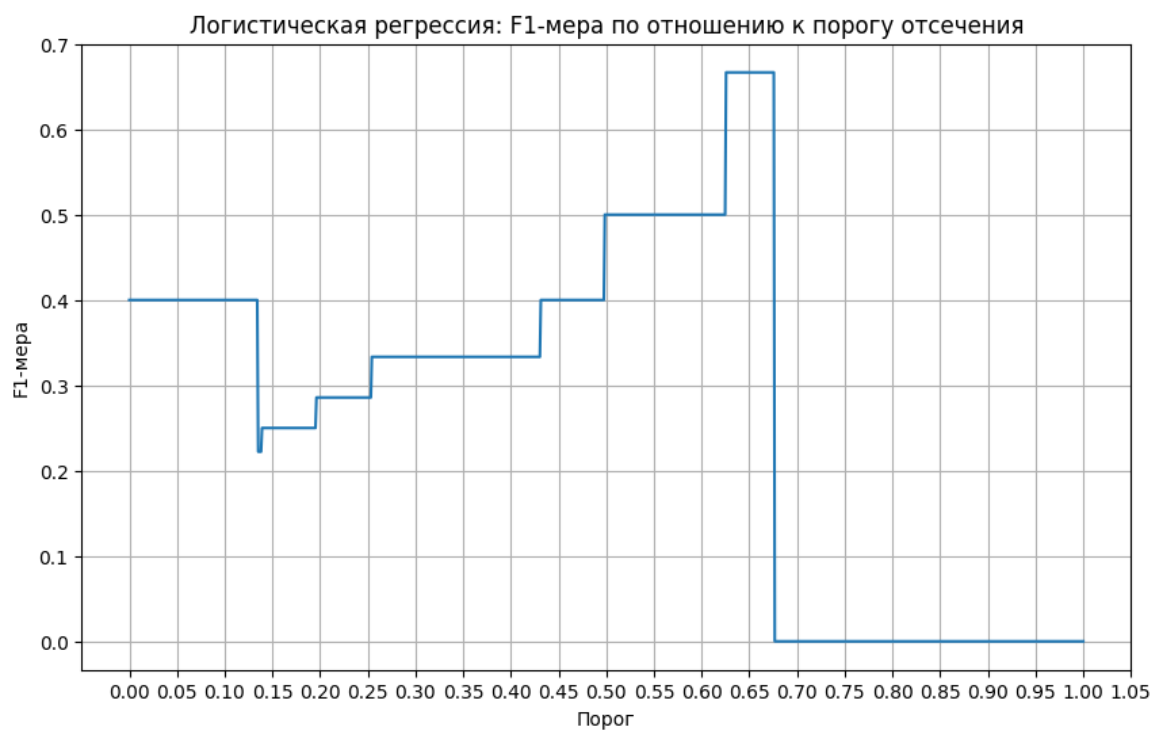


Рисунок 16 — график для KNN

Визуально определим  $\text{threshold} = 0.55$ .



Визуально определим  $\text{threshold} = 0.65$ .

Рисунок 17 — график для логистической регрессии

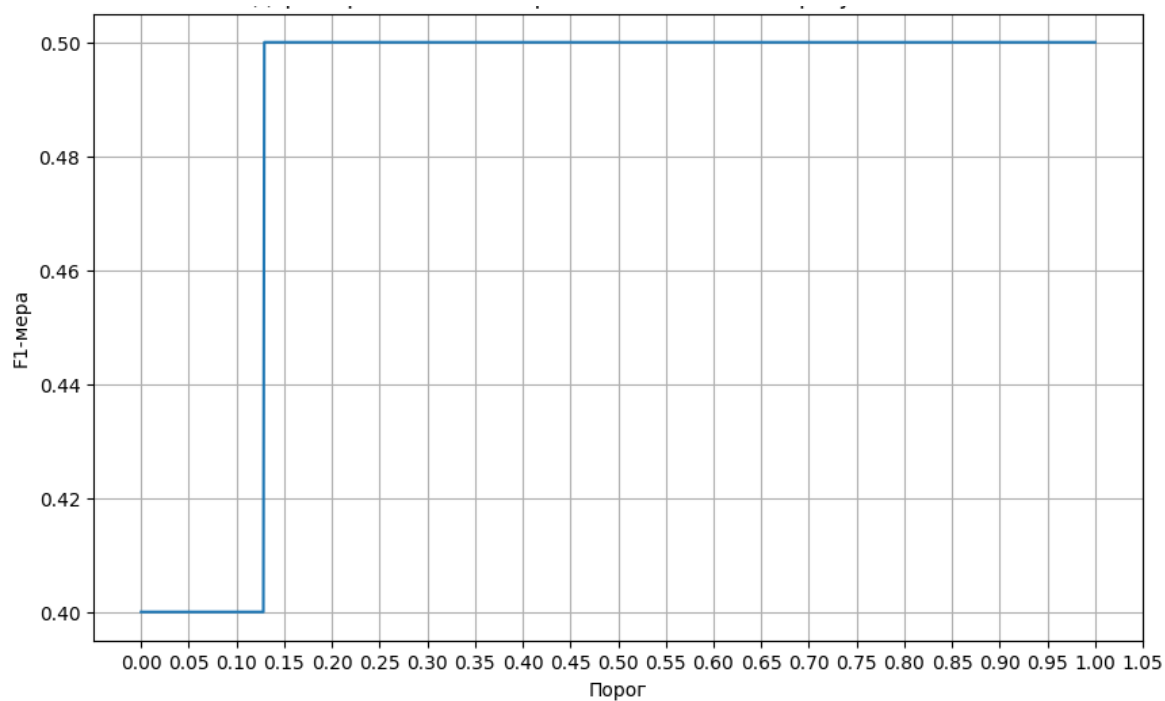


Рисунок 18 — график для дерева решений

Визуально определим  $\text{threshold} = 0.3$ .

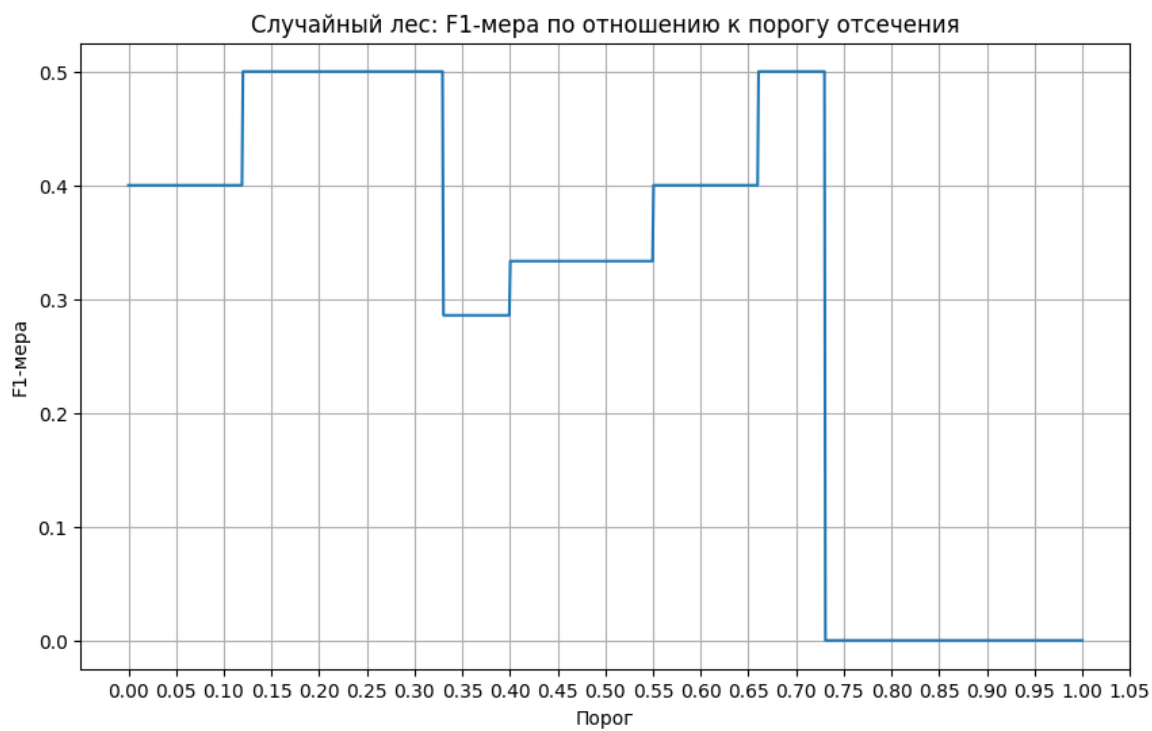


Рисунок 19 — график для случайного леса

Визуально определим  $\text{threshold} = 0.7$ .

Построим таблицу полученных метрик для каждой из модели на тренировочных данных.

	KNN	ЛГ	ДР	СЛ
Accuracy	0.75	0.875	0.75	0.75
Precision	0.5	1.0	0.5	0.5
Recall	0.5	0.5	0.5	0.5
F1	0.5	0.666	0.5	0.5
ROC-AUC	0.666	0.75	0.66	0.666

Построим таблицу полученных метрик для каждой из модели на тестовых данных.

	KNN	ЛГ	ДР	СЛ
Accuracy	0.769	0.794	0.897	0.974
Precision	1.0	0.833	1.0	1.0
Recall	0.25	0.416	0.666	0.916
F1	0.4	0.556	0.8	0.956
ROC-AUC	0.65	0.69	0.833	0.958

Сравним полученные метрики. Введем следующие обозначения.

- + — метрика на тестовых данных оказалась выше
- — метрика на тестовых данных оказалась ниже
- = — значения совпали

	KNN	ЛГ	ДР	СЛ
Accuracy	+	-	+	+
Precision	+	-	+	+

Recall	-	-	+	+
F1	-	-	+	+
ROC-AUC	=	-	+	+

Как можно заметить, переобучение явно выражено в модели логистической регрессии, так как все метрики уменьшились в значениях по сравнению с обучающими данными.

Более высокие результаты показали методы случайного леса и дерева решений.

**Задание 8. Сравнить полученную важность признаков в модели логистической регрессии, в модели деревьев решений и в случайном лесу (для древесных моделей это можно сделать с помощью ключа `feature_importances` у обученной модели). Проинтерпретировать полученную важность признаков.**

Важность признаков логистической регрессии определяется весами. Положительные веса означают положительное влияние признака.

Важность признаков в модели логистической регрессии

```

      x2      x3      x4      x5      x6      x7      x8 \
0 -0.612072  0.813513 -0.619968  0.391649 -0.608739 -1.211684 -0.550841

      x9
0  0.311528

```

Рисунок 20 — веса логистической регрессии

Согласно рисунку 20, положительное влияние в логистической регрессии имеют признаки  $x_3$ ,  $x_5$ ,  $x_9$ , где

- $x_3$  – число разводов на 1000 человек;
- $x_5$  – к-т младенческой смертности;
- $x_9$  – числа зарегистрированных преступлений на 100000 населения.



Для модели дерева решений важность признаков показывает насколько хорошо разделение по этому признаку улучшает качество прогнозов. Так, в модели ДР важными стали признаки  $x_2$ ,  $x_4$ ,  $x_9$ , где

- $x_2$  – число смертности населения на 1000 человек;
- $x_4$  – число разводов на 1000 человек;
- $x_9$  – числа зарегистрированных преступлений на 100000 населения.

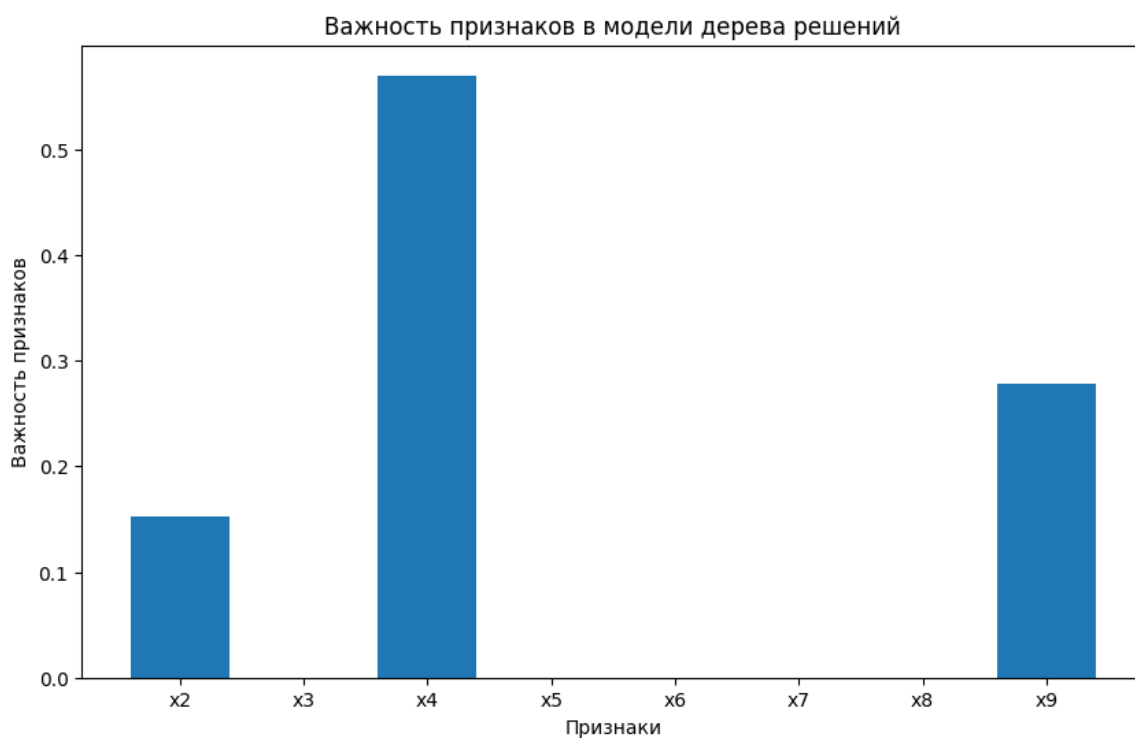


Рисунок 21 — важность признаков модели дерева решений

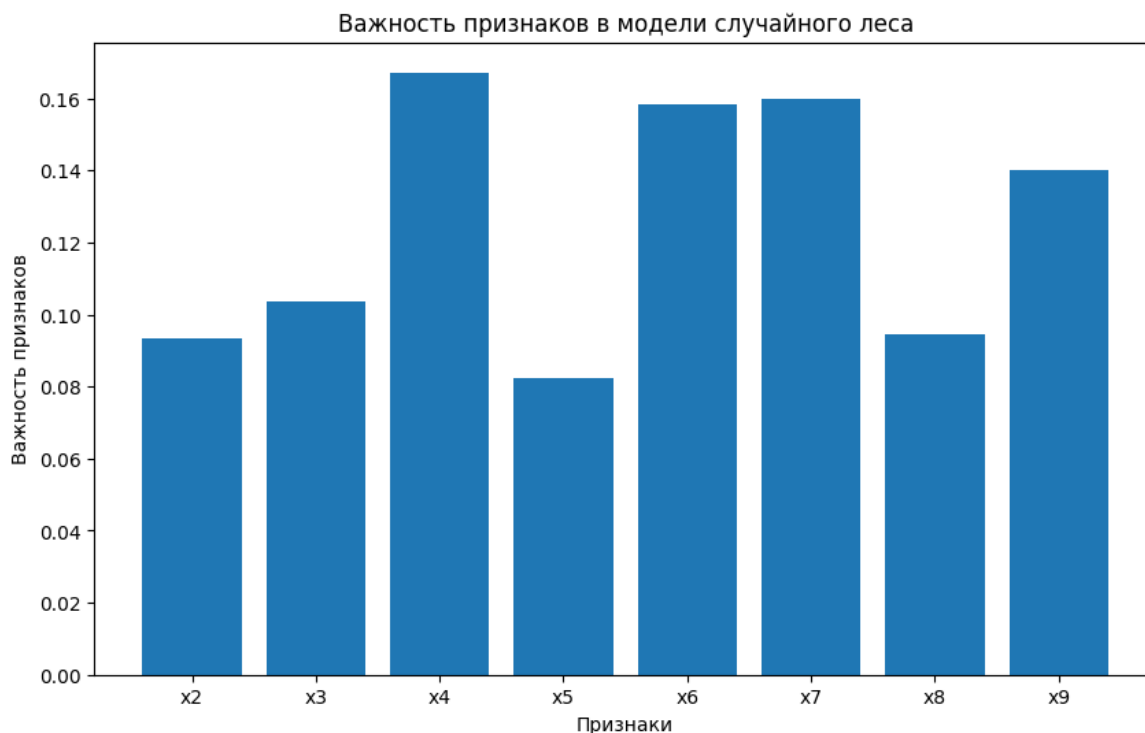


Рисунок 22 — важность признаков модели случайного дерева

Для модели случайного леса оказались важными большинство признаков.

Таким образом, можно сказать что дерево решений использует только три признака, в то время как СЛ и ЛГ использует все признаки в той или иной степени.

### **Вывод**

Изучила такие методы обучения, как KNN, логистическая регрессия, дерево решений и случайный лес, а также основные метрики для анализа результатов моделей.