

## Постановка задачи:

Требуется построить и протестировать классификатор многомерных объектов на основе дискриминантного анализа при наличии обучающей выборки: а) из модельных данных б) из репозитория

### а) выборка из модельных данных

---

#### Построение дискриминантной функции по обучающей выборке, классификация тестовой выборки

Бинарный классификатор должен быть протестирован для двух случаев: хорошо и плохо разделенные данные, распределенные по закону многомерного нормального распределения размерности  $p = 3$ .

Для каждого случая по отдельности должны быть заданы различные векторы средних и равные матрицы ковариаций. Хорошо и плохо разделенные данные будем выбирать, меняя матрицу ковариаций.

Хорошо разделенные данные:

ОВ1:

$$x \sim N(\mu^{(1)}, \Sigma_{good})$$

ОВ2:

$$x \sim N(\mu^{(2)}, \Sigma_{good})$$

Плохо разделенные данные:

ОВ1:

$$x \sim N(\mu^{(1)}, \Sigma_{bad})$$

ОВ2:

$$x \sim N(\mu^{(2)}, \Sigma_{bad})$$

---

Подключим необходимые для исследования пакеты:

```
In [1]: from scipy.stats import multivariate_normal, norm
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Создадим набор необходимых функций для построения классификатора. Для большего удобства функции были "упакованы" в класс **MyClassifier**.

Основной для классификации является функция **calc\_estimates**, в ней рассчитываются оценки средних

$$\mu^{(k)} \rightarrow \hat{\mu}^{(k)}, \quad \hat{\mu}_j^{(k)} = \frac{1}{n_k} \sum_{i=1}^{n_k} x_{ij}, \quad k = 1, 2$$

и оценка матрицы ковариаций

$$S^{(k)} = (s_{lj}^{(k)}), \quad l, j = \overline{1, 3}, k = 1, 2$$

$$s_{lj}^{(k)} = \frac{1}{n_k - 1} \sum_{i=1}^{n_k} (x_{il}^{(k)} - \hat{\mu}_l^{(k)})(x_{ij}^{(k)} - \hat{\mu}_j^{(k)}), \quad k = 1, 2$$

$$\Sigma \rightarrow S, \quad S = \frac{1}{n_1 + n_2 - 2} [(n_1 - 1)S^{(1)} + (n_2 - 1)S^{(2)}]$$

оценка вектора  $\alpha$ :

$$\alpha \rightarrow \hat{\alpha} = \mathbf{a}, \quad \alpha = \Sigma^{-1}(\mu^{(1)} - \mu^{(2)}) \rightarrow \mathbf{a} = S^{-1}(\hat{\mu}^{(1)} - \hat{\mu}^{(2)})$$

а также оценки средних дискриминантной функции

$$\xi_k \rightarrow \bar{z}^{(k)} = \langle \hat{\mu}^{(k)}, \mathbf{a} \rangle, \quad k = 1, 2$$

и дисперсии

$$\sigma_z^2 \rightarrow s_z^2 = \sum_{l=1}^3 \sum_{j=1}^3 a_l s_{lj} a_j$$

```
In [4]: class MyClassifier():
    def __init__(self, mean1, mean2, cov, q1, size, test_set=None):
        self.sample1 = []
        self.sample2 = []
        self.m1 = mean1
        self.m2 = mean2
        self.cov = cov
        self.q1 = q1
        self.q2 = 1 - q1
        self.n1 = int(size * q1)
        self.n2 = size - self.n1
        self.test_set = test_set

        self.m1_ = [] # Выборочная оценка
        self.m2_ = [] # Выборочная оценка
        self.cov_ = [] # Выборочная оценка
        self.alpha = []

        self.mz1_ = 0 # средние z (их полусумма используется для определения порога)
        self.mz2_ = 0
        self.z_var = 0

    def generate_sample(self):
        self.sample1 = multivariate_normal(mean=self.m1, cov=self.cov).rvs(s
        self.sample2 = multivariate_normal(mean=self.m2, cov=self.cov).rvs(s

    def calc_estimates(self):
```

```

#           samples mean
self.m1_ = np.mean(self.sample1, axis=0)
self.m2_ = np.mean(self.sample2, axis=0)
# default normalization with N - 1
self.cov_ = (np.cov(self.sample1.T) + np.cov(self.sample2.T)) / 2

self.alpha = np.linalg.inv(self.cov_) @ (self.m1_ - self.m2_)
#           mean of discriminant function
self.mz1_ = np.dot(self.alpha, self.m1_)
self.mz2_ = np.dot(self.alpha, self.m2_)
#           variance of discriminant function
self.z_var = self.alpha @ self.cov_ @ self.alpha

def mahalanobis(self, unbiased=False):
    makh = (self.mz1_ - self.mz2_) ** 2 / self.z_var
    if unbiased:
        p = len(self.m1)
        makh = ((self.n1 + self.n2 - p - 3) /
                (self.n1 + self.n2 - 2)) * makh - p * (1 / self.n1 + 1 /
    return makh

def calc_errors(self, D):
    K = np.log(self.q2 / self.q1)
    F = lambda x: norm.cdf(x)
    return {"p21": F((K - 0.5 * D ** 2) / D), "p12": F((-K - 0.5 * D **

def specify_test_set(self, create_new=False):
    if create_new:
        print("Creating test set")
        test1 = multivariate_normal(mean=self.m1, cov=self.cov).rvs(self
        test2 = multivariate_normal(mean=self.m2, cov=self.cov).rvs(self
    else:
        test1 = self.sample1
        test2 = self.sample2
    self.test_set = np.vstack([test1, test2])
    return np.hstack([np.zeros(self.n1), np.ones(self.n2)])

# Возвращает массив ответов-предсказаний
def predict(self, test_set=None):
    if not test_set:
        test_set = self.test_set
    if test_set is None:
        print("Warning!\n\tTest set not specified!")
        return
    #           (\k_{s1} + \k_{s2}) / 2
    threshold = (self.mz1_ + self.mz2_) / 2
    lnq1q2 = np.log(self.q2 / self.q1)
    threshold += lnq1q2
    # Массив ответов (предсказаний)
    predict = []
    for instance in test_set:
        if np.dot(self.alpha, instance) >= threshold:
            predict.append(0)
        else:
            predict.append(1)
    return predict

```

Зададим размер обучающей выборки  $n_1 = 1000$ ,  $n_2 = 1000$ , векторы средних  $m_1$  и  $m_2$ , матрицы ковариаций для хорошо и плохо разделимых данных -  $s_{good}$  и  $s_{bad}$  соответственно.

```
In [5]: n = 2000

m1 = np.array([4, 5, 7])
m2 = np.array([1, 9, 2])

s_good = np.array([[1.2, 0, -1],
                   [0, 3, 0.5],
                   [-1, 0.5, 4]])

s_bad = (s_good + 1.5) * 5

q1 = 0.5
```

Создадим классы исследований. Сгенерируем обучающие выборки и тестовые выборки. Размер тестовой выборки  $n_T = 2000$ , выборка содержит равное число элементов из 1 и 2 классов.

```
In [6]: R_good = MyClassifier(m1, m2, s_good, q1, n)
R_bad = MyClassifier(m1, m2, s_bad, q1, n)
R_good.generate_sample()
R_bad.generate_sample()
true_good = R_good.specify_test_set(create_new=True)
true_bad = R_bad.specify_test_set(create_new=True)
```

Creating test set  
Creating test set

Рассчитаем необходимые для классификатора величины.

```
In [7]: R_good.calc_estimates()
R_bad.calc_estimates()
```

Получим предсказания следующим образом:

Если

$$\sum_{j=1}^p a_j x_j = \langle \mathbf{a}, \mathbf{x} \rangle \geq \frac{\bar{z}^{(1)} + \bar{z}^{(2)}}{2} + \ln \frac{q_2}{q_1}$$

то относим экземпляр тестовой выборки к 1 классу - иначе к 2 классу.

$q_1, q_2$  - относительная частота 1 и 2 класса в обучающей выборке. В данном случае  $q_1 = q_2 = 0.5$ .

```
In [8]: predict_good = R_good.predict()
predict_bad = R_bad.predict()
```

Рассчитаем эмпирическую вероятность ошибочной классификации:

$$P_3(1|2) = \frac{m_2}{n_2}$$

$m_2$  - количество элементов 2 класса, которых классификатор определил как 1 класс

$$P_3(2|1) = \frac{m_1}{n_1}$$

$m_1$  - количество элементов 1 класса, которых классификатор определил как 2 класс

И построим четырехпольную таблицу сопряженности.

```
In [9]: cm_good_test = confusion_matrix(true_good, predict_good)
cm_bad_test = confusion_matrix(true_bad, predict_bad)

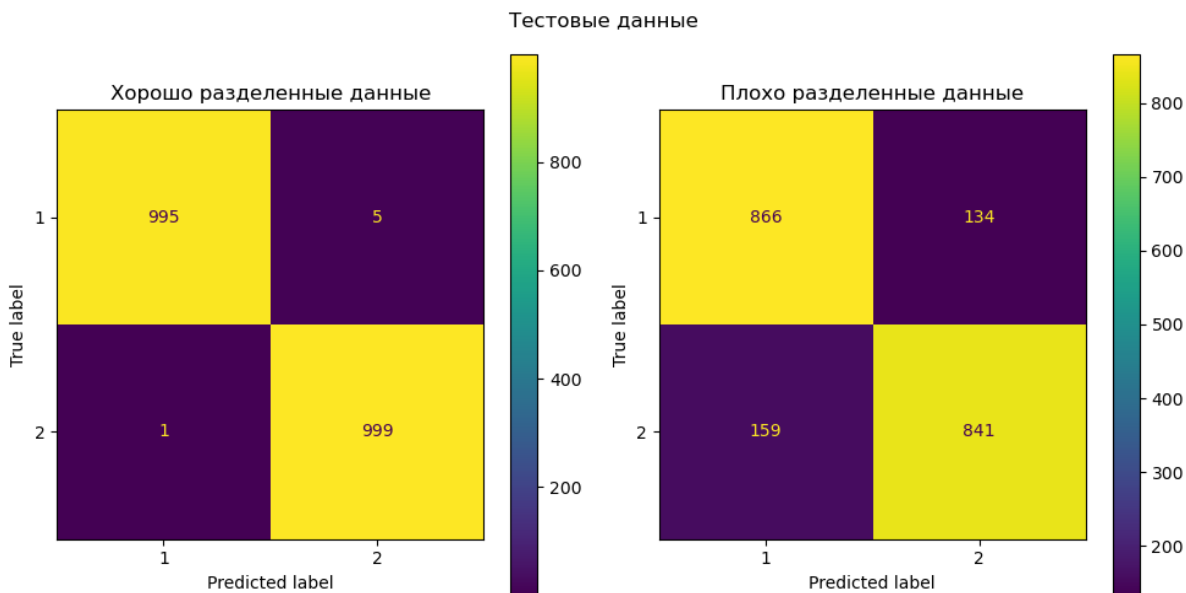
def confusion_matrix_report(cm_good, cm_bad, subtitle: str, err_name: str, R
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ConfusionMatrixDisplay(cm_good, display_labels=["1", "2"]).plot(ax=ax[0])
ax[0].set_title("Хорошо разделенные данные")
ConfusionMatrixDisplay(cm_bad, display_labels=["1", "2"]).plot(ax=ax[1])
ax[1].set_title("Плохо разделенные данные")
fig.suptitle(subtitle)
fig.tight_layout()
plt.show()

print(err_name + ":",

_, m21, m12, _ = cm_good.ravel()
print(f"\tХорошо разделенные данные: \n\t\tP(1|2) = {m12 / R_good.n2}, P(2|1)

_, m21, m12, _ = cm_bad.ravel()
print(f"\tПлохо разделенные данные: \n\t\tP(1|2) = {m12 / R_bad.n2}, P(2|1)

confusion_matrix_report(cm_good_test, cm_bad_test, subtitle='Тестовые данные',
err_name='Эмпирическая вероятность ошибочной классификации')
```



Эмпирическая вероятность ошибочной классификации:

Хорошо разделенные данные:

$$P(1|2) = 0.001, P(2|1) = 0.005$$

Плохо разделенные данные:

$$P(1|2) = 0.159, P(2|1) = 0.134$$

## Исследование классификации исходной обучающей выборки

Укажем тестовую выборку (без аргумента *create\_new=True* в методе

**specify\_test\_set()** в качестве классифицируемых указываются исходные

обучающие выборки)

```
In [10]: true_good = R_good.specify_test_set()
true_bad = R_bad.specify_test_set()
predict_good = R_good.predict()

predict_bad = R_bad.predict()
```

Рассчитаем оценки вероятностей ошибочной классификации:

$$\hat{P}(1|2) = \frac{m_2}{n_2}$$

$m_2$  - количество элементов 2 класса, которых классификатор определил как 1 класс

$$\hat{P}(2|1) = \frac{m_1}{n_1}$$

$m_1$  - количество элементов 1 класса, которых классификатор определил как 2 класс

И построим четырехпольную таблицу сопряженности.

**А также (чтобы было удобно сравнивать) вновь выведем четырехпольную таблицу и эмпирические вероятности по итогам классификации тестовой выборки.**

С той же целью сразу выведем несмещенную оценку расстояния Махаланобиса:

$$D_{\text{H}}^2 = \frac{n_1 + n_2 - p - 3}{n_1 + n_2 - 2} D^2 - p \left( \frac{1}{n_1} + \frac{1}{n_2} \right), \quad p = 3$$

где

$$D^2 = \frac{(\bar{z}^{(1)} - \bar{z}^{(2)})^2}{s_z^2}$$

И оценки вероятностей ошибочной классификации для каждого класса:

$$\hat{P}(2|1) = \Phi \left( \frac{K - \frac{1}{2} D_{\text{H}}^2}{D_{\text{H}}} \right)$$

$$\hat{P}(1|2) = \Phi \left( \frac{-K - \frac{1}{2} D_{\text{H}}^2}{D_{\text{H}}} \right)$$

$$\Phi(y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^y e^{-\frac{t^2}{2}} dt$$

$$K = \ln \left( \frac{q_2 c(1|2)}{q_1 c(2|1)} \right)$$

Где  $c(1|2)$ ,  $c(2|1)$  - стоимости ошибочной классификации

В нашем случае  $q_1 = q_2 = 0.5$ , а  $c(1|2) = c(2|1)$  так как задача не предполагает выделения важности конкретной ошибки. Следовательно  $K = 0$ .

Также посчитаем вероятность ошибочной классификации, которая минимизируется в ходе Байесовской процедуры:

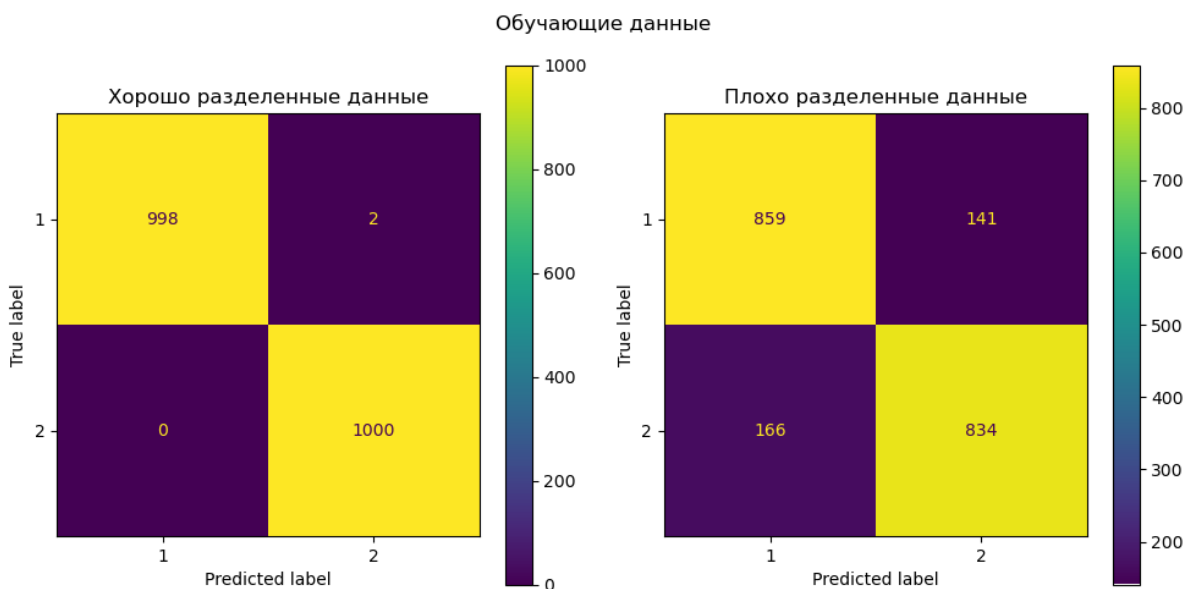
$$q_1 \cdot \hat{P}(2|1) + q_2 \cdot \hat{P}(1|2)$$

```
In [11]: cm_good = confusion_matrix(true_good, predict_good)
cm_bad = confusion_matrix(true_bad, predict_bad)

confusion_matrix_report(cm_good, cm_bad, subtitle='Обучающие данные',
                        err_name='Оценка вероятности ошибочной классификации', R_g
confusion_matrix_report(cm_good_test, cm_bad_test, subtitle='Тестовые данные',
                        err_name='Эмпирическая вероятность ошибочной классификации'

print()

for header, research in zip(["Хорошо разделенные данные:", "Плохо разделенные даннь
[R_good, R_bad]):
    print(header)
    makh = research.makhalanobis(unbiased=True)
    errors = research.calc_errors(makh)
    print(f"\tНесмещенная оценка расстояния Махаланобиса: {round(makh, 3)}", )
    print(f"\tОценки ошибочной классификации: P(2|1) = {round(errors['p21'], 3)}")
    print(f"\tВероятность ошибочной классификации: {round(research.q1 * errors['p21
```



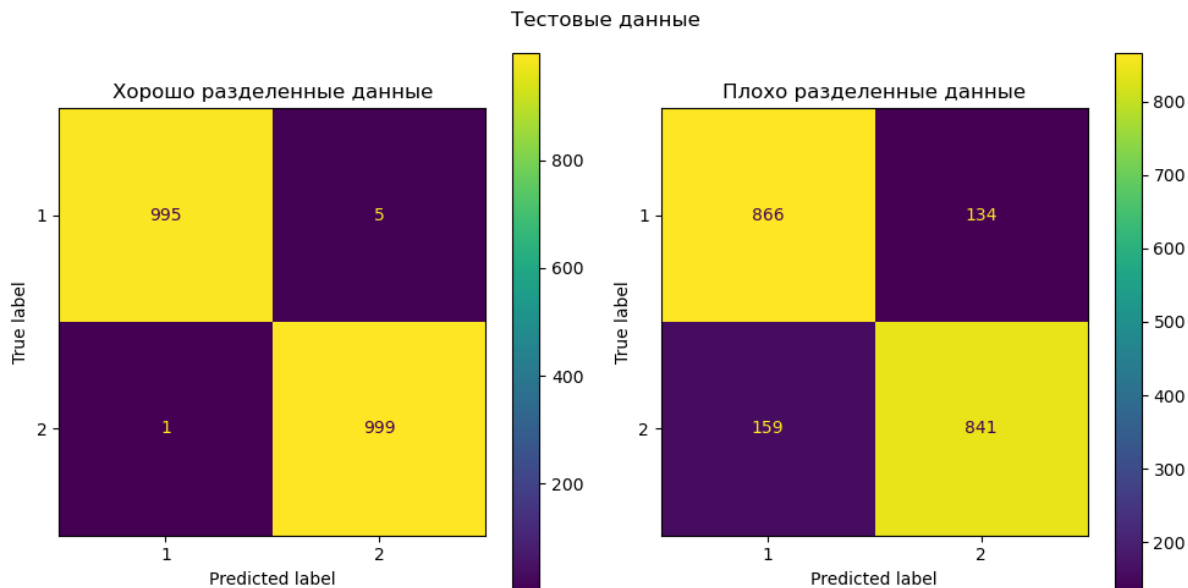
Оценка вероятности ошибочной классификации:

Хорошо разделенные данные:

$$P(1|2) = 0.0, P(2|1) = 0.002$$

Плохо разделенные данные:

$$P(1|2) = 0.166, P(2|1) = 0.141$$



Эмпирическая вероятность ошибочной классификации:

Хорошо разделенные данные:

$$P(1|2) = 0.001, P(2|1) = 0.005$$

Плохо разделенные данные:

$$P(1|2) = 0.159, P(2|1) = 0.134$$

Хорошо разделенные данные:

Несмещенная оценка расстояния Махаланобиса: 5.984

Оценки ошибочной классификации:  $P(2|1) = 0.001$

$$P(1|2) = 0.001$$

Вероятность ошибочной классификации: 0.001

Плохо разделенные данные:

Несмещенная оценка расстояния Махаланобиса: 2.102

Оценки ошибочной классификации:  $P(2|1) = 0.147$

$$P(1|2) = 0.147$$

Вероятность ошибочной классификации: 0.147

## Сравнительный анализ

Заметим, что и для тестовой выборки, и для обучающих данных, классификация хорошо разделенных данных получается качественнее - эмпирическая вероятность ошибочной классификации и ее оценка в обоих случаях ниже нежели в случае плохо разделенных данных.

Что и требовалось ожидать - расстояние Махаланобиса меньше, а оценки вероятностей ошибочной классификации больше в случае плохо разделенных данных.

## б) выборка из данных репозитория

### Построение дискриминантной функции по обучающей выборке, классификация тестовой выборки

Исследуем качество классификации на данных датасета *german-numeric*. Размер датасета - 1000, 700 из них принадлежит классу №1, 300 - №2. Число признаков - 24.

```
In [14]: class CreditClassifier:
def __init__(self, x, y, train_coef):
```



```

logging = True
self.X = x
self.Y = y

self.sample1 = self.X[self.Y == 1]
self.sample2 = self.X[self.Y == 2]

size1 = len(self.sample1)
size2 = len(self.sample2)
if logging:
    print(f"Всего данных: \t\t1: {size1} \t2: {size2}")

self.train1 = self.sample1[:int(train_coef * size1)]
self.train2 = self.sample2[:int(train_coef * size2)]
self.train_trueY = np.hstack([np.ones(len(self.train1)), np.full(len(self.train2), 2)])
self.train_set = np.vstack([self.train1, self.train2])
self.train_n1 = len(self.train1)
self.train_n2 = len(self.train2)

self.test1 = self.sample1[int(train_coef * size1):]
self.test2 = self.sample2[int(train_coef * size2):]
# true targets
self.test_trueY = np.hstack([np.ones(len(self.test1)), np.full(len(self.test2), 2)])
self.test_set = np.vstack([self.test1, self.test2])
self.test_n1 = len(self.test1)
self.test_n2 = len(self.test2)

if logging:
    print(f"Тренировочные данные: \t1: {len(self.train1)} \t2: {len(self.train2)}")
    print(f"Тестовые данные: \t1: {len(self.test1)} \t2: {len(self.test2)}")

self.q1 = len(self.train1) / (len(self.train1) + len(self.train2))
self.q2 = 1 - self.q1

if logging:
    print(f"q1 = {self.q1}, \tq2 = {round(self.q2, 2)}")

def calc_estimates(self):
    # samples mean
    self.m1_ = np.mean(self.train1, axis=0)
    self.m2_ = np.mean(self.train2, axis=0)

    # specify covariance matrix for samples
    # assuming that cov1 approximately equal cov2
    self.cov_ = ((len(self.train1) - 1) * np.cov(self.train1.T) + (len(self.train2) - 1) * np.cov(self.train2.T)) / (len(self.train1) + len(self.train2) - 2)

    self.alpha = np.linalg.inv(self.cov_) @ (self.m1_ - self.m2_)
    # mean of discriminant function
    self.mz1_ = np.dot(self.alpha, self.m1_)
    self.mz2_ = np.dot(self.alpha, self.m2_)
    # variance of discriminant function
    self.z_var = self.alpha @ self.cov_ @ self.alpha

def mahalanobis(self, unbiased=False):
    makh = (self.mz1_ - self.mz2_) ** 2 / self.z_var
    if unbiased:
        p = len(self.m1_)
        n1 = self.train_n1
        n2 = self.train_n2
        makh = ((n1 + n2 - p - 3) / (n1 + n2 - 2)) * makh - p * (1 / n1 + 1 / n2)
    return makh

def calc_errors(self, D):

```

```

K = np.log(self.q2 / self.q1)
F = lambda x: norm.cdf(x)
return {"p21": F((K - 0.5 * D ** 2) / D), "p12": F((-K - 0.5 * D **

def predict(self, predict_test_set=True):
    if predict_test_set:
        predict_set = self.test_set
    else:
        predict_set = self.train_set
    threshold = (self.mz1_ + self.mz2_) / 2
    lnq1q2 = np.log(self.q2 / self.q1)
    threshold += lnq1q2
    predict = []
    for instance in predict_set:
        if np.dot(self.alpha, instance) >= threshold:
            predict.append(1)
        else:
            predict.append(2)
    return predict

```

Для обучения берем 90% данных датасета. 630 от 1-го класса, 270 от 2-го.

```

In [15]: X = []
Y = []
with open("german.data-numeric") as data:
    for string in data:
        string = string.split()
        x, y = string[:-1], string[-1]
        X.append(np.array(list(map(lambda elem: float(elem), x))))
        Y.append(int(y))
X = np.array(X)
Y = np.array(Y)

repo_classificator = CreditClassificator(X, Y, 0.9)
repo_classificator.calc_estimates()

```

```

Всего данных:          1: 700  2: 300
Тренировочные данные:  1: 630  2: 270
Тестовые данные:      1: 70   2: 30
q1 = 0.7,              q2 = 0.3

```

Найдем эмпирическую вероятность ошибочной классификации.

```

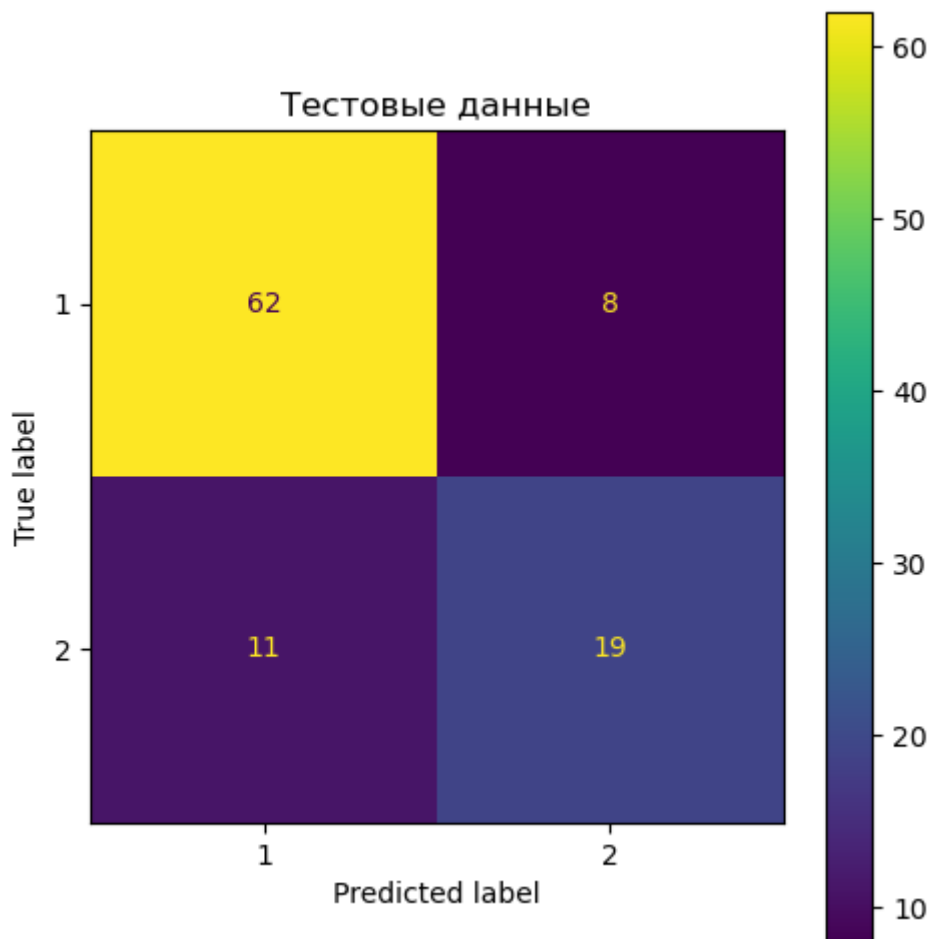
In [16]: predict = repo_classificator.predict(predict_test_set=True)

cm_test = confusion_matrix(repo_classificator.test_trueY, predict)

fig, ax = plt.subplots(1, 1, figsize=(5, 5))
ConfusionMatrixDisplay(cm_test, display_labels=["1", "2"]).plot(ax=ax)
ax.set_title("Тестовые данные")
fig.tight_layout()
plt.show()

_, m21, m12, _ = cm_test.ravel()
p12_test, p21_test = round(m12 / repo_classificator.test_n2, 3), round(m21 / r
print(
    f"Эмпирические вероятности ошибочной классификации: \n\tP(1|2) = {round(m12 / r

```



Эмпирические вероятности ошибочной классификации:

$$P(1|2) = 0.367, P(2|1) = 0.114$$

## Исследование классификации исходной обучающей выборки из репозитория

Вычислим оценки вероятностей ошибочной классификации, а также несмещенную оценку расстояние Махаланобиса, оценки вероятностей ошибочной классификации и вероятность совокупной ошибки классификации по ранее упомянутым [формулам 17-22](#).

Также для удобства сравнения выведем рядом четырехпольную таблицу сопряженности и эмпирические вероятности по итогам классификации тестовой выборки.

```
In [17]: predict = repo_classifier.predict(predict_test_set=False)

cm = confusion_matrix(repo_classifier.train_trueY, predict)

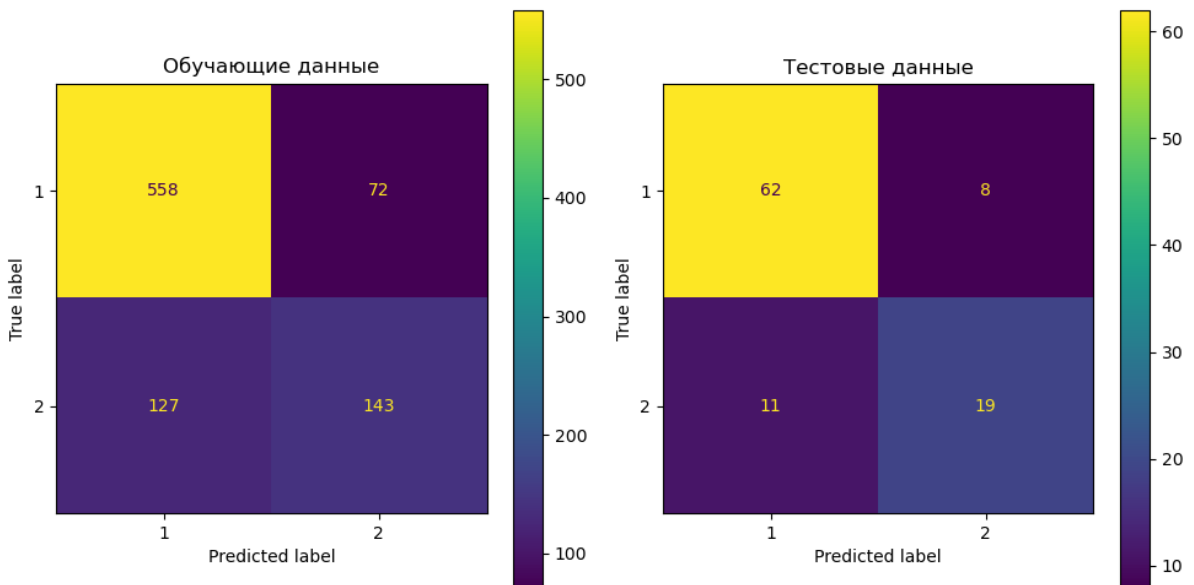
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ConfusionMatrixDisplay(cm, display_labels=["1", "2"]).plot(ax=ax[0])
ax[0].set_title("Обучающие данные")
ConfusionMatrixDisplay(cm_test, display_labels=["1", "2"]).plot(ax=ax[1])
ax[1].set_title("Тестовые данные")
fig.tight_layout()
plt.show()

_, m21, m12, _ = cm.ravel()
```

```

print(
    f"Оценки вероятностей ошибочной классификации: \n\tP(1|2) = {round(m12 / repo_
    makh = repo_classifier.makhalanobis(unbiased=True)
    errors = repo_classifier.calc_errors(makh)
    print(f"Несмещенная оценка расстояния Махаланобиса: {round(makh, 3)}")
    print(f"Оценки ошибочной классификации: P(1|2) = {round(errors['p12'], 3)} \t P(
    print(
        f"Вероятность ошибочной классификации: {round(repo_classifier.q1 * errors[
    print()
    print(f"Эмпирические вероятности ошибочной классификации: \n\tP(1|2) = {p12_test},

```



Оценки вероятностей ошибочной классификации:

$$P(1|2) = 0.47, \quad P(2|1) = 0.114$$

Несмещенная оценка расстояния Махаланобиса: 1.198

Оценки ошибочной классификации:  $P(1|2) = 0.543$

$$P(2|1) = 0.096$$

Вероятность ошибочной классификации: 0.23

Эмпирические вероятности ошибочной классификации:

$$P(1|2) = 0.367, \quad P(2|1) = 0.114$$

Чтобы сразу перейти к отчету по классификации после понижения размерности нажмите [здесь](#)

## Метод главных компонент

### Исследование классификации тестовой выборки из репозитория с уменьшенной размерностью данных (PCA)

Попробуем предобработать данные. Сократим число признаков с помощью метода главных компонент.

Стандартизуем данные, вычислим несмещенную оценку матрицы ковариаций. Вычислим собственные значения и векторы. Отберем наиболее значимые признаки, домножив матрицу признаков на матрицу состоящую из собственных векторов, собственные числа которых удовлетворяют неравенству (правило Кайзера):

$$\lambda_i : \frac{\lambda_i}{\sum_{j=1}^{p=24} D[x_j]} = \frac{\lambda_i}{\text{trace}(S)} > \frac{1}{24}$$

Так как данные были стандартизированы, то отбор собственных чисел становится следующим:

$$\lambda_i : \lambda_i > 1$$

```
In [18]: X = []
Y = []
with open("german.data-numeric") as data:
    for string in data:
        string = string.split()
        x, y = string[:-1], string[-1]
        X.append(np.array(list(map(lambda elem: float(elem), x))))
        Y.append(int(y))
X = np.array(X)
Y = np.array(Y)

mean = np.mean(X, axis=0)
covariance = np.cov(X.T)
X_centered = X - mean
X_stand = X_centered / np.sqrt(np.diagonal(covariance))

covariance = np.cov(X_stand.T)

eiges = np.linalg.eig(covariance)

values, vectors = eiges
eiges = sorted(list(zip(values, vectors.T)), key=lambda x: x[0], reverse=True)

vals = values[values > 1]
print(f"Отобрали следующие {len(vals)} собственных чисел: {vals}")
```

```
Отобрали следующие 10 собственных чисел: [2.51828975 2.12147659 1.85393992 1.703
97873 1.63217357 1.32147521
1.21521006 1.15970619 1.12146973 1.01054463]
```

Далее произведем перемножение матриц:

$$X[1000, 24] \cdot Transform[24, 10] = X_{new}[1000, 10]$$

Числа в скобках отображают размерности матриц. Матрица  $Transform[24, 10]$  состоит из собственных векторов, соответствующих отобраным собственным значениям.

```
In [30]: transform_ = np.vstack(list([eiges[i][1] for i in range(10)]))
X_new = X_stand @ transform_.T + mean @ transform_.T
```

```
In [31]: credit_classifier_PCA = CreditClassifier(X_new, Y, 0.9)
credit_classifier_PCA.calc_estimates()
```

```
Всего данных:          1: 700   2: 300
Тренировочные данные:  1: 630   2: 270
Тестовые данные:      1: 70    2: 30
q1 = 0.7,              q2 = 0.3
```

```
In [32]: predict = credit_classifier_PCA.predict(predict_test_set=True)
```

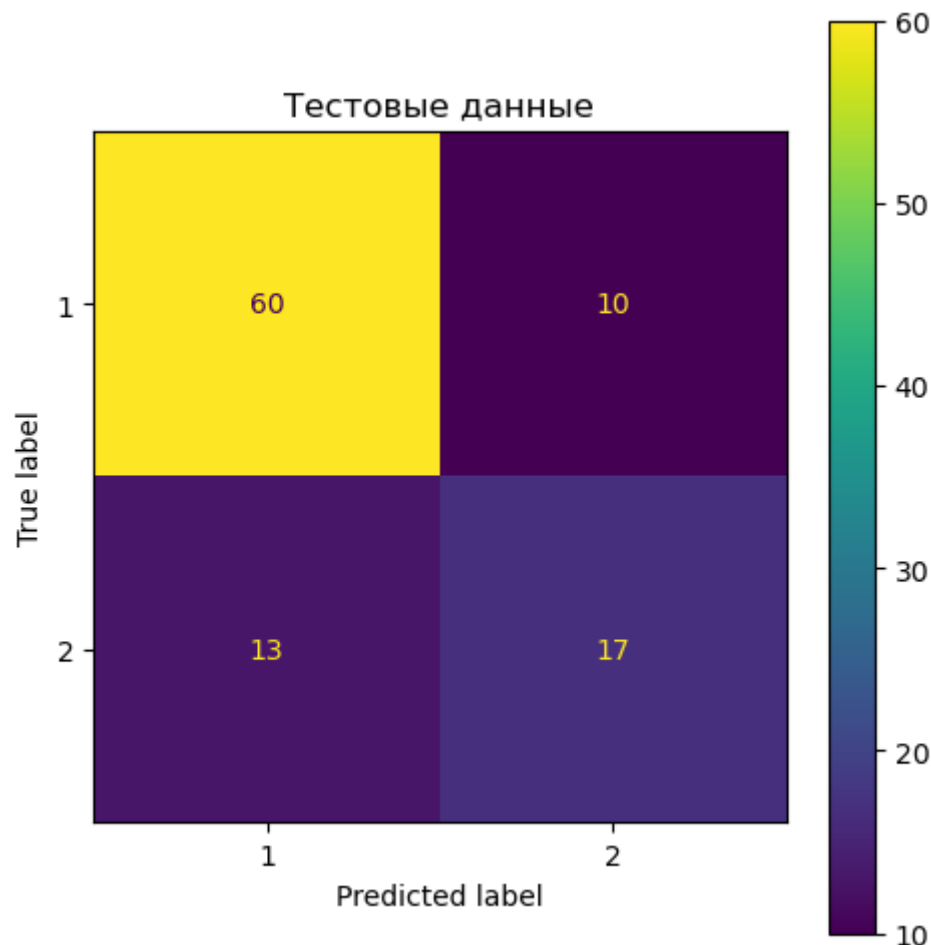
```

cm_test_pca = confusion_matrix(credit_classifier_PCA.test_trueY, predict)

fig, ax = plt.subplots(1, 1, figsize=(5, 5))
ConfusionMatrixDisplay(cm_test_pca, display_labels=["1", "2"]).plot(ax=ax)
ax.set_title("Тестовые данные")
fig.tight_layout()
plt.show()

_, m21, m12, _ = cm_test_pca.ravel()
p12_test_pca, p21_test_pca = round(m12 / credit_classifier_PCA.test_n2, 3)
print(f"Эмпирические вероятности ошибочной классификации: \n\tP(1|2) = {p12_test_pca}

```



Эмпирические вероятности ошибочной классификации:

$$P(1|2) = 0.433, P(2|1) = 0.143$$

## Исследование классификации исходной обучающей выборки из репозитория с уменьшенной размерностью данных (PCA)

```

In [33]: predict = credit_classifier_PCA.predict(predict_test_set=False)

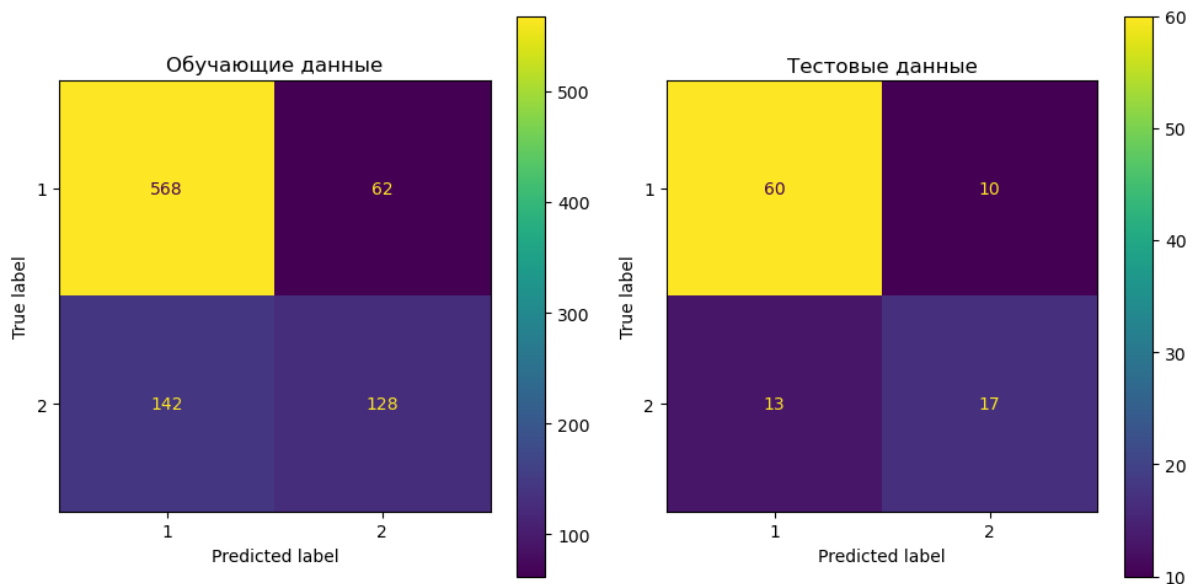
cm = confusion_matrix(credit_classifier_PCA.train_trueY, predict)

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ConfusionMatrixDisplay(cm, display_labels=["1", "2"]).plot(ax=ax[0])
ax[0].set_title("Обучающие данные")
ConfusionMatrixDisplay(cm_test_pca, display_labels=["1", "2"]).plot(ax=ax[1])
ax[1].set_title("Тестовые данные")
fig.tight_layout()
plt.show()

_, m21, m12, _ = cm.ravel()

```

```
print(
    f"Оценки вероятностей ошибочной классификации: \n\tP(1|2) = {round(m12 / credi
print(f"Эмпирические вероятности ошибочной классификации: \n\tP(1|2) = {p12_test_pc
```



Оценки вероятностей ошибочной классификации:

$$P(1|2) = 0.526, P(2|1) = 0.098$$

Эмпирические вероятности ошибочной классификации:

$$P(1|2) = 0.433, P(2|1) = 0.143$$

```
In [34]: makh = credit_classifier_PCA.makhalanobis(unbiased=True)
errors = credit_classifier_PCA.calc_errors(makh)
print(f"\tНесмещенная оценка расстояния Махаланобиса: {round(makh, 3)}")
print(f"\tОценки ошибочной классификации: P(1|2) = {round(errors['p12'], 3)} \t
print(
    f"\tВероятность ошибочной классификации: {round(credit_classifier_PCA.q1 *

Несмещенная оценка расстояния Махаланобиса: 1.14
Оценки ошибочной классификации: P(1|2) = 0.569      P(2|1) = 0.095
Вероятность ошибочной классификации: 0.237
```

## Выводы по PCA

Понижение размерности может уменьшить сложность вычислений, также можно использовать его для экономии трафика (если мы передаем эти признаки от клиента к серверу). Но уменьшив число признаков более чем в два раза, пришлось пожертвовать точностью разделения объектов по классам.

В данном случае вероятность ошибки второго класса приближается к 0.5, что создает сомнения в смысле использования такого классификатора - случайный выбор мог определять второй класс с тем же успехом.

Опираясь на совокупную вероятность ошибочной классификации, можем сделать вывод, что наш классификатор все же работает лучше случайной классификации. Вероятность ошибочной классификации, вычисленная по [формулам 17-23](#), в случае исходных обучающих данных составляет 0.23, после понижения размерности ошибка незначительно возросла и стала равной 0.237.

Отметим, что  $P_2(2|1)$ , и  $\hat{P}(2|1)$  маленькие, если в контексте задачи критически важным было бы не допускать ошибку отнесения 1го класса к 2му, то наш

классификатор бы хорошо справился (например, 1 класс - наличие опухоли, а 2 - отсутствие, лучше перестраховаться и сказать, что опухоль есть, чтобы снимок посмотрел врач).

## Заключение

---

- Убедились, что при плохо разделенных данных классификация с помощью дискриминантного анализа работает хуже
- Создали классификаторы на основе разных выборок, и подсчитали вероятности ошибочной классификации
- Понизили размерность векторов признаков с помощью метода главных компонент, несущественно потеряв в точности