Отчёт по лабораторной работе №2 "Логистическая регрессия. Многоклассовая классификация"

In [1]:

```
import os
import random
import numpy as np
import pandas as pd
import scipy.optimize as opt
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import fmin, fmin_bfgs
np.seterr(all='ignore')

DATA_FILE_NAME_1 = 'Lab 2/ex2data1'
DATA_FILE_NAME_2 = 'Lab 2/ex2data2'
DATA_FILE_NAME_3 = 'Lab 2/ex2data3'
```

Imports from 'common' file:

In [2]:

```
import os
from scipy.io import loadmat

DATA_DIRECTORY = '../Data/'


def load_data(filename, convert_type=float, separator=',', directory=DATA_DIRECTORY, encoding='utf-8',
              skip_extention=False, split_function=None):

    filepath = directory + filename
    if not skip_extention:
        filepath += '.txt'

    if not split_function:
        def split_function(line):
            return line.replace('\n', '').split(separator)

    data = []
    with open(filepath, 'r', encoding=encoding) as f:
        for line in f.readlines():
            try:
                if isinstance(convert_type, (list, tuple)):
                    data.append([
                        convert_type[j](el)
                        for row in line.replace('\n', '')
                        for j, el in enumerate(row.split(separator))
                    ])
                else:
                    data.append([convert_type(x) for x in split_function(line)])
            except TypeError:
                pass

    return np.array(data)

def load_data_from_mat_file(filename, directory=DATA_DIRECTORY):
    filepath = directory + f'{filename}.mat'
    return loadmat(filepath)
```

1. Загрузите данные *ex2data1.txt* из текстового файла.

In [3]:

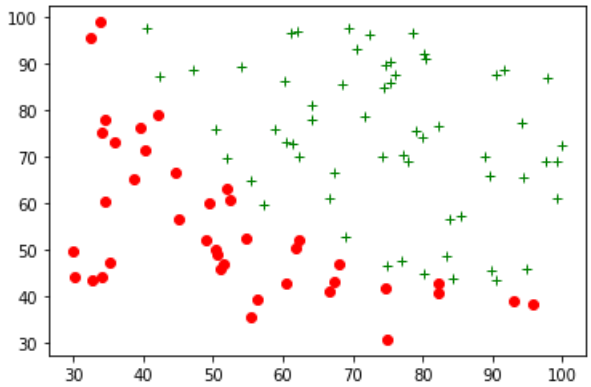
```
data_array = load_data(DATA_FILE_NAME_1)
```

```
df_1 = pd.DataFrame({'ex1': data_array[:, 0], 'ex2': data_array[:, 1], 'passed': data_array[:, 2]})
sorted_df_1 = df_1.sort_values(by=['ex1', 'ex2'])
```

2. Постройте график, где по осям откладываются оценки по предметам, а точки обозначаются двумя разными маркерами в зависимости от того, поступил ли данный студент в университет или нет.

In [4]:

```
dataset_passed = sorted_df_1.query('passed==1')
plt.plot(dataset_passed.ex1, dataset_passed.ex2, 'g+')
dataset_not_passed = sorted_df_1.query('passed==0')
plt.plot(dataset_not_passed.ex1, dataset_not_passed.ex2, 'ro')
plt.show()
```



3. Реализуйте функции потерь J(θ) и градиентного спуска для логистической регрессии с использованием векторизации.

In [5]:

```
X = sorted_df_1[['ex1', 'ex2']].values
X = np.concatenate((np.ones((len(X), 1)), X), axis=1)
Y = sorted_df_1['passed'].values


def logistic_regration_hypotesis(T, X):
    return 1 / (1 + np.exp(-X.dot(T)))


def cost_function(H, Y):
    return -(Y.dot(np.log(H)) + (np.ones(Y.shape) - Y).dot(np.log(np.ones(H.shape) - H))) / len(Y)


def get_cost_function(X, Y):
    def func(T):
        H = logistic_regration_hypotesis(T, X)
        return cost_function(H, Y)
    return func


def gradient_function(X, Y, H):
    return (H - Y).dot(X) / len(Y)


def get_gradient_function(X, Y):
    def func(T):
        H = logistic_regration_hypotesis(T, X)
        return gradient_function(X, Y, H)
    return func


def gradient_descent_function(T, X, Y, learning_rate=0.1, eps=1e-3, iteration_count=1e5):
    iteration = 0
    theta_gradient = [eps + 1]
    _gradient_function = get_gradient_function(X, Y)
```

```
    while np.any(np.abs(theta_gradient) > eps) and iteration < iteration_count:
        theta_gradient = _gradient_function(T)
        T -= learning_rate * theta_gradient
        iteration += 1

    return T, not np.any(np.abs(theta_gradient) > eps), iteration


T0 = np.array([-2, 0.1, 0.1])
eps = 1.0e-6

theta, success, iteration = gradient_descent_function(T0, X, Y, learning_rate=4*1e-3, eps=eps, iter
ation_count=1e6)
print('Custom GD:')
print('theta', theta)
print('success', success)
print('iteration', iteration, '\n')
```

```
Custom GD:
theta [-24.71253321   0.20264203   0.19784   ]
success False
iteration 1000000
```

4. Реализуйте другие методы (как минимум 2) оптимизации для реализованной функции стоимости (например, Метод Нелдера — Мида, Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно, генетические методы и т.п.). Разрешается использовать библиотечные реализации методов оптимизации (например, из библиотеки scipy).

Реализуем метод Недлера-Мида:

In [6]:

```
print('Nedler-Mead with regularization:')
T0 = np.array([-2, 0.1, 0.1])
eps = 1.0e-6

cost_func = get_cost_function(X, Y)
res = opt.minimize(cost_func, T0, method='Nelder-Mead', options={'xtol': eps, 'disp': True})

print('success:', res.success)
print('x:', res.x)
print('\n\n')
```

```
Nedler-Mead with regularization:
Optimization terminated successfully.
         Current function value: 0.203498
         Iterations: 196
         Function evaluations: 349
success: True
x: [-25.16133377   0.20623171   0.2014716 ]
```

Реализуем метод BFGS:

In [7]:

```
print('BFGS with regularization:')
T0 = np.array([-2, 0.1, 0.1])

cost_func = get_cost_function(X, Y)
grad_func = get_gradient_function(X, Y)
res = opt.minimize(cost_func, T0, method='bfgs', jac=grad_func, options={'disp': True})

print('success:', res.success)
print('x:', res.x)
print('\n\n')
```

```
BFGS with regularization:
Optimization terminated successfully.
         Current function value: 0.203498
         Iterations: 25
         Function evaluations: 29
         Gradient evaluations: 29
success: True
x: [-25.16134055   0.20623176   0.20147166]
```

Как можно видеть из ответов, все методы минимазации функции стоимости дают приблизительно одинаковые результаты.

5. Реализуйте функцию предсказания вероятности поступления студента в зависимости от значений оценок по экзаменам.

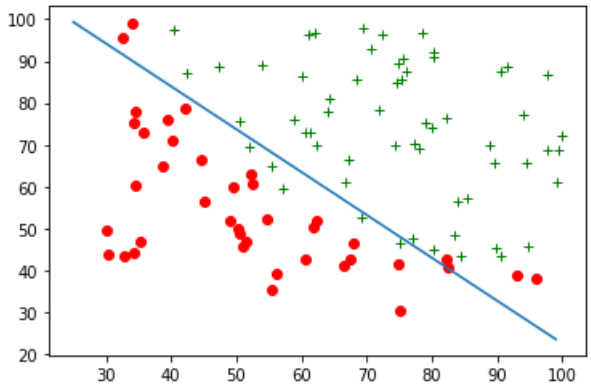
In [8]:

```
def predict(T, X):
    prediction = logistic_regration_hypotesis(T, X)
    return 1 if prediction >= 0.5 else 0
```

6. Постройте разделяющую прямую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 2.

In [9]:

```
T_opt = res.x

def decision_boundary_function(T, x):
    return (-T[0] - T[1] * x) / T[2]


x = [*range(25, 100)]
H = [decision_boundary_function(T_opt, xi) for xi in x]

plt.plot(dataset_passed.ex1, dataset_passed.ex2, 'g+')
plt.plot(dataset_not_passed.ex1, dataset_not_passed.ex2, 'ro')
plt.plot(x, H)
plt.show()
```



7. Загрузите данные ex2data2.txt из текстового файла.

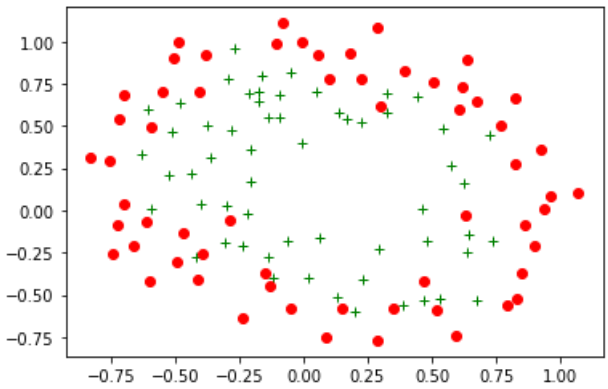
In [10]:

```
data_array = load_data(DATA_FILE_NAME_2)
df_2 = pd.DataFrame({'test1': data_array[:, 0], 'test2': data_array[:, 1], 'passed': data_array[:,
2]})
sorted_df_2 = df_2.sort_values(by=['test1', 'test2'])
X = sorted_df_2[['test1', 'test2']].values
Y = sorted_df_2['passed'].values
```

8. Постройте график, где по осям откладываются результаты тестов, а точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет.

In [11]:

```
dataset_passed = sorted_df_2.query('passed==1')
plt.plot(dataset_passed.test1, dataset_passed.test2, 'g+')
dataset_not_passed = sorted_df_2.query('passed==0')
plt.plot(dataset_not_passed.test1, dataset_not_passed.test2, 'ro')
plt.show()
```



9. Постройте все возможные комбинации признаков x1 (результат первого теста) и x2 (результат второго теста), в которых степень полинома не превышает 6, т.е. 1, x1, x2, x1 ^ 2, x1 *x2, x2 ^ 2, ..., x1* x2 ^ 5, x2 ^ 6 (всего 28 комбинаций).

In [12]:

```
def get_extended_x(X):
    extended_X = []

    for i in range(0, 7):
        for j in range(0, 7):
            if i + j > 6:
                continue

            extended_X.append([x[0] ** i * x[1] ** j for x in X])

    return np.array(extended_X).T


E_X = get_extended_x(X)
print(E_X.shape)
```

```
(118, 28)
```

10.Реализуйте L2-регуляризацию для логистической регрессии и обучите ее на расширенном наборе признаков методом градиентного спуска.

In [13]:

```
def cost_function_with_reg(H, Y, R):
    return cost_function(H, Y) + R.sum() / len(Y)


def get_cost_function_with_reg(X, Y, reg_param):
    def func(T):
        H = logistic_regration_hypotesis(T, X)
        R = np.square(T[1:]) * reg_param
        return cost_function_with_reg(H, Y, R)
    return func
```

```
def gradient_function_with_reg(X, Y, H, R):
    return gradient_function(X, Y, H) + R / len(Y)


def get_gradient_function_with_reg(X, Y, reg_param):
    def func(T):
        H = logistic_regration_hypotesis(T, X)
        R = T * reg_param
        R[0] = 0
        return gradient_function_with_reg(X, Y, H, R)
    return func


def gradient_descent_function_with_reg(T, X, Y, learning_rate, reg_param, eps, iteration_count):
    iteration = 0
    theta_gradient = [eps + 1]
    _gradient_function = get_gradient_function_with_reg(X, Y, reg_param)

    while np.any(np.abs(theta_gradient) > eps) and iteration < iteration_count:
        theta_gradient = _gradient_function(T)
        T -= learning_rate * theta_gradient
        iteration += 1

    return T, not np.any(np.abs(theta_gradient) > eps), iteration


T0 = np.zeros(28, dtype=float)

learning_rate = 1.0e-1
reg_param = 1e-2
eps = 1.0e-4
iteration_count = 1.0e6
l2_theta, success, iteration = gradient_descent_function_with_reg(
    T0, E_X, Y, learning_rate, reg_param, eps, iteration_count)

cost_func = get_cost_function_with_reg(E_X, Y, reg_param)

print('L2 regularization:')
print('Current function value:', cost_func(l2_theta))
print('iteration:', iteration)
print('success:', success)
print('x:', list(l2_theta), '\n')
```

```
L2 regularization:
Current function value: 0.3471333630999231
iteration: 132282
success: True
x: [3.821749367715735, 4.631182186085838, -6.496415643500933, -2.3171682744081634, -
5.469485232304071, 2.3212984169512887, 0.22967898199512685, 2.1230617074395073, -
6.570779300620059, 2.041853246251168, -2.222739641882225, -3.571468840767241, -3.1696208372958012,
-5.5194010018709765, -0.5146378629673963, -3.4377186735900724, -3.3236589467068973, -
3.7284423407053695, 1.9569185757840923, 2.922258481419257, 4.30943900338022, 2.560782624187576, -4
.128693280951285, -0.5862939961561395, -0.8719464339549892, -1.5380161654338464,
0.9691207725303475, -5.004436366617219]
```

11.Реализуйте другие методы оптимизации.

Реализуем метод Недлера-Мида:

In [14]:

```
T0 = np.ones(28, dtype=float)
reg_param = 1e-2
eps = 1e-4

nm_res = fmin(cost_func, T0, xtol=eps, maxfun=1e7)
print('x:', res)
print('\n\n')
```

```
Optimization terminated successfully.
         Current function value: 0.415550
```

```
         Iterations: 20023
         Function evaluations: 24144
x:       fun: 0.20349770158946018
 hess_inv: array([[ 3.18931259e+03, -2.56664777e+01, -2.57712144e+01],
       [-2.56664777e+01,  2.21489281e-01,  1.93743986e-01],
       [-2.57712144e+01,  1.93743986e-01,  2.24550961e-01]])
      jac: array([-3.99981806e-08, -2.61317887e-06, -2.10333225e-06])
  message: 'Optimization terminated successfully.'
     nfev: 29
      nit: 25
     njev: 29
   status: 0
  success: True
        x: array([-25.16134055,   0.20623176,   0.20147166])
```

Реализуем метод BFGS:

In [15]:

```
T0 = np.ones(28, dtype=float)
reg_param = 1e-2
eps = 1e-4

grad_func = get_gradient_function_with_reg(E_X, Y, reg_param)
bfgs_res = fmin_bfgs(cost_func, T0, fprime=grad_func, gtol=eps, disp=True)
print('x:', res)
print('\n\n')
```

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.351805
         Iterations: 53
         Function evaluations: 97
         Gradient evaluations: 85
x:       fun: 0.20349770158946018
 hess_inv: array([[ 3.18931259e+03, -2.56664777e+01, -2.57712144e+01],
       [-2.56664777e+01,  2.21489281e-01,  1.93743986e-01],
       [-2.57712144e+01,  1.93743986e-01,  2.24550961e-01]])
      jac: array([-3.99981806e-08, -2.61317887e-06, -2.10333225e-06])
  message: 'Optimization terminated successfully.'
     nfev: 29
      nit: 25
     njev: 29
   status: 0
  success: True
        x: array([-25.16134055,   0.20623176,   0.20147166])
```

Не сложно заметить, что результаты для всех алгоритмов получились довольно схожими

12.Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.

Функция предсказания будет идентичной методу predict, реализованному в 5-м пункте.

In [16]:

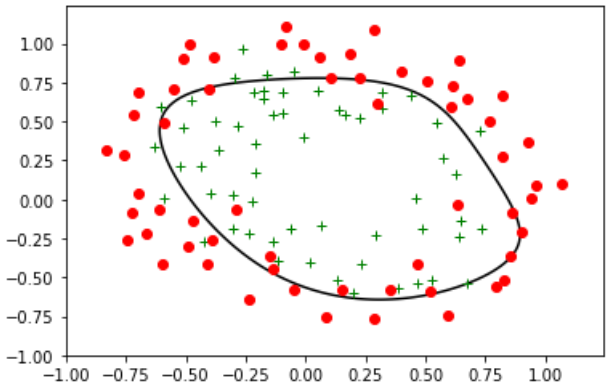
```
x, actual = E_X[15], Y[15]

print('Custom L2 regularisation:')
print(f'Predicted: {predict(l2_theta, x)} Actual: {actual}')
print('Nedler-Mead optimization:')
print(f'Predicted: {predict(nm_res, x)} Actual: {actual}')
print('BFGS optimization:')
print(f'Predicted: {predict(bfgs_res, x)} Actual: {actual}')
```

```
Custom L2 regularisation:
Predicted: 1 Actual: 1.0
Nedler-Mead optimization:
Predicted: 1 Actual: 1.0
BFGS optimization:
Predicted: 1 Actual: 1.0
```

13.Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.

In [17]:

```
T = l2_theta
ex_x = []
X1 = np.arange(-1., 1.25, 0.01)
X2 = np.arange(-1., 1.25, 0.01)

e_x = get_extended_x([[x1, x2] for x1 in X1 for x2 in X2])
Z = np.array([logistic_regration_hypotesis(T, e_x)])


X1, X2 = np.meshgrid(X1, X2)
Z = Z.reshape(X1.shape)
plt.contour(X1, X2, Z, colors='black', levels=1)
plt.plot(dataset_passed.test1, dataset_passed.test2, 'g+')
plt.plot(dataset_not_passed.test1, dataset_not_passed.test2, 'ro')

plt.show()
```



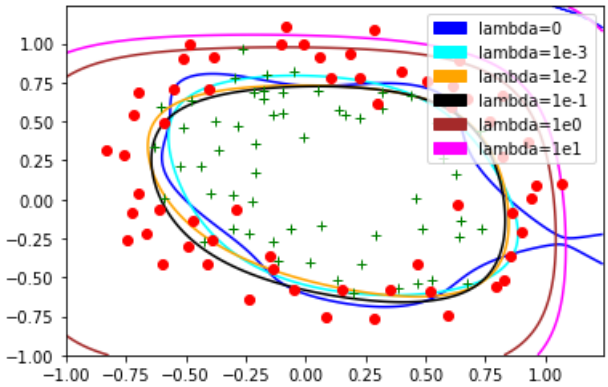
14.Попробуйте различные значения параметра регуляризации λ. Как выбор данного значения влияет на вид разделяющей кривой? Ответ дайте в виде графиков.

In [18]:

```
T0 = np.ones(28, dtype=float)
success = False
reg_params = [0] + [1e-2 * 10 ** i for i in range(5)]
Ts = []

for reg_param in reg_params:
    cost_func = get_cost_function_with_reg(E_X, Y, reg_param)
    grad_func = get_gradient_function_with_reg(E_X, Y, reg_param)
    res = fmin_bfgs(cost_func, T0, fprime=grad_func, gtol=1e-4, disp=True)
    Ts.append(res)

contours = []
colors = ['blue', 'cyan', 'orange', 'black', 'brown', 'magenta']
X1 = np.arange(-1., 1.25, 0.01)
X2 = np.arange(-1., 1.25, 0.01)

e_x = get_extended_x([[x1, x2] for x1 in X1 for x2 in X2])
for i, t in enumerate(Ts):
    Z = np.array([logistic_regration_hypotesis(t, e_x)])

    x1, x2 = np.meshgrid(X1, X2)
    Z = Z.reshape(x1.shape)
```

```
    contours.append(plt.contour(x1, x2, Z, levels=1, colors=colors[i]))

plt.plot(dataset_passed.test1, dataset_passed.test2, 'g+')
plt.plot(dataset_not_passed.test1, dataset_not_passed.test2, 'ro')
plt.legend(
    [mpatches.Patch(color=colors[i]) for i, v in enumerate(reg_params)],
    (['lambda=0'] + [f'lambda=1e{i-3}' for i in range(len(reg_params) - 1)])
)

plt.show()
```

```
Optimization terminated successfully.
         Current function value: 0.274309
         Iterations: 261
         Function evaluations: 263
         Gradient evaluations: 263
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.351805
         Iterations: 53
         Function evaluations: 97
         Gradient evaluations: 85
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.440442
         Iterations: 21
         Function evaluations: 130
         Gradient evaluations: 118
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.596087
         Iterations: 19
         Function evaluations: 79
         Gradient evaluations: 68
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.678857
         Iterations: 13
         Function evaluations: 107
         Gradient evaluations: 95
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 0.692452
         Iterations: 6
         Function evaluations: 98
         Gradient evaluations: 86
```



In [19]:

```
reg_param0 = np.array([1e-2 * 10 ** i for i in range(5)] + [0])
reg_param0.sort()
print(reg_param0)
```

```
[0.e+00 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02]
```

Как видно из графика, увеличение параметра регуляризации λ приводит к увеличению области решений, также к снижению ошибки обобщения, т.е. уменьшает вероятность переобучения. При λ=0 можем заметить дольно точное разделение классов, что ведет к переобучению.

15.Загрузите данные ex2data3.mat из файла.

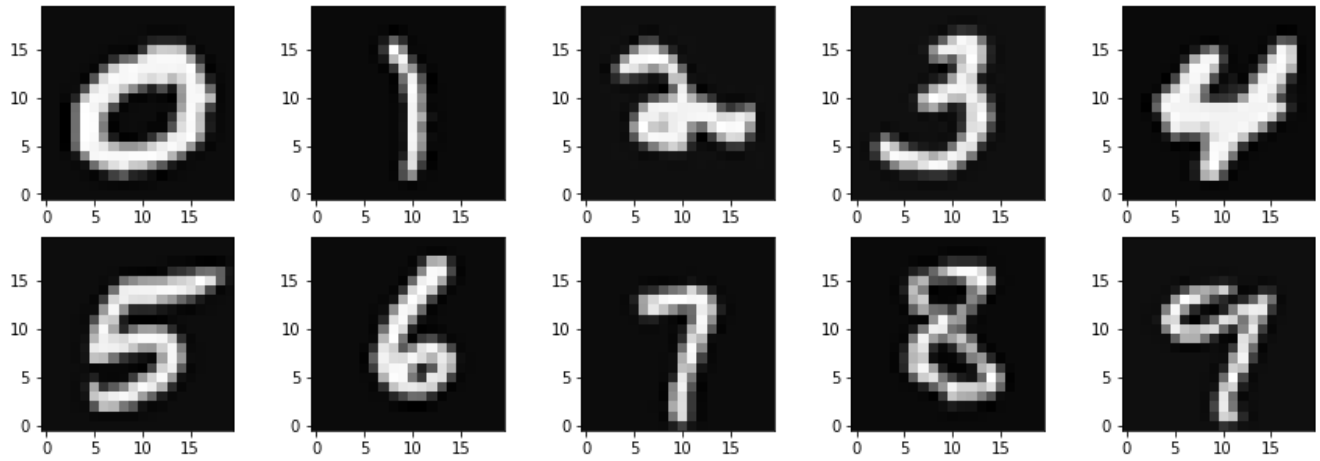
In [114]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_3)
X = df_data['X']
Y = df_data['y'][:, 0]

df_3 = pd.DataFrame(
    np.concatenate((df_data['X'], df_data['y']), axis=1),
    columns=[*[f'x{i}' for i in range(400)], 'y'])
```

16.Визуализируйте несколько случайных изображений из набора данных. Визуализация должна содержать каждую цифру как минимум один раз.

In [21]:

```
images_hash = {}
for i, v in enumerate(X):
    k = Y[i]
    if k not in images_hash:
        images_hash[k] = [v]
        continue

    images_hash[k].append(v)

shape = (20, 20)
fig, axs = plt.subplots(2, 5, figsize=(15, 5))
axs = axs.flatten()
for k, v in images_hash.items():
    ax = axs[k % 10]
    image = random.choice(v)
    data = image.reshape(shape).T
    data = np.flip(data, axis=0)
    im = ax.imshow(data, cmap='gray', origin='lower')

plt.show()
```



17.Реализуйте бинарный классификатор с помощью логистической регрессии с использованием векторизации (функции потерь и градиентного спуска).

In [22]:

```
def get_binary_classificator(y0, X, Y):
    T0 = np.ones(len(X[0]), dtype=float)
    y = [1 if y == y0 else 0 for y in Y]
    iteration_count = 1e4
    learning_rate = 1e-1
    eps = 1e-1

    theta, success, iteration = gradient_descent_function(T0, X, y, learning_rate, eps,
iteration_count)
```

```
    if not success:
        print('Binary classification with L2 regularization:')
        print('success', success)
        print('iteration', iteration, '\n')

    def func(x):
        print(logistic_regration_hypotesis(theta, x))
        return predict(theta, x)

    return func


classificator = get_binary_classificator(3, X, Y)
print(classificator(X[1503]))
```

```
0.6698350118734239
1
```

18.Добавьте L2-регуляризацию к модели.

In [28]:

```
def get_binary_classificator_with_reg(y0, X, Y, lr=1e-1, reg_param=1e2, eps = 1e-3, iteration_count
= 1e5):
    T0 = np.zeros(len(X[0]), dtype=float)
    y = [1 if y == y0 else 0 for y in Y]

    theta, success, iteration = gradient_descent_function_with_reg(
        T0, X, y, lr, reg_param, eps, iteration_count)

    if not success:
        print('Binary classification with L2 regularization:')
        print('success', success)
        print('iteration', iteration, '\n')

    def func(x):
        return predict(theta, x)

    return func


classificator = get_binary_classificator_with_reg(3, X, Y)
print(classificator(X[1503]))
```

```
1
```

19.Реализуйте многоклассовую классификацию по методу “один против всех”.

In [39]:

```
shuffled_df_3 = df_3.sample(frac=1)
unique_y = np.unique(df_3['y'])

tranning = shuffled_df_3[:int(len(shuffled_df_3) * 0.8)]
tranning_x = tranning[tranning.columns.difference(['y'])].values
tranning_y = tranning['y'].values

classificators = {v: get_binary_classificator_with_reg(v, tranning_x, tranning_y) for v in unique_y
}


def get_multiclass_classification(x, classificators=classificators):
    predictions = [classificator(x) for c in classificators]
    index = np.argmax(predictions)
    return np.eye(len(classificator), k=index)
```

20.Реализуйте функцию предсказания класса по изображению с использованием обученных классификаторов.

In [30]:

```
testing = shuffled_df_3[int(len(shuffled_df_3) * 0.8):]
testing_x = testing[testing.columns.difference(['y'])].values
testing_y = testing['y'].values


def predict_class(X, classificators=classificators):
    probs = {y_class: classificator(X) for y_class, classificator in classificators.items()}
    return max(probs, key=probs.get)  # class number

for index in np.random.randint(0, high=len(testing_x), size = 3):
    print(f'Prediction: {predict_class(testing_x[index])}, Real: {testing_y[index]}')
```

```
Prediction: 1.0, Real: 7.0
Prediction: 8.0, Real: 8.0
Prediction: 2.0, Real: 2.0
```

Не соответствия классов произошло по причине неправильного подбора коэфициентов обучения классификаторов. Для поверышения точности стоит уменьшить пороговую ошибку (eps), а также уменьшить коэфициент регуляции - обобщения(reg_param).

21.Процент правильных классификаций на обучающей выборке должен составлять около 95%.

In [112]:

```
params = {
    'lr': 2,
    'eps': 1e-4,
    'reg_param': 0.1,
    'iteration_count': 1e5,
}
new_classificators = {v: get_binary_classificator_with_reg(v, tranning_x, tranning_y, **params) for
v in unique_y}
```

In [113]:

```
true_predictions = 0
for index, row in tranning.iterrows():
    predicted_class = predict_class(row[tranning.columns.difference(['y'])], classificators=new_cla
ssificators)
    if predicted_class == row['y']:
        true_predictions += 1

print('prediction accuracy:', true_predictions / len(tranning))
```

```
prediction accuracy: 0.90125
```