Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе:

Лабораторная работа №2 “Логистическая регрессия”

Выполнил: Карп Александр Игоревич
магистрант кафедры информатики
группа №858641

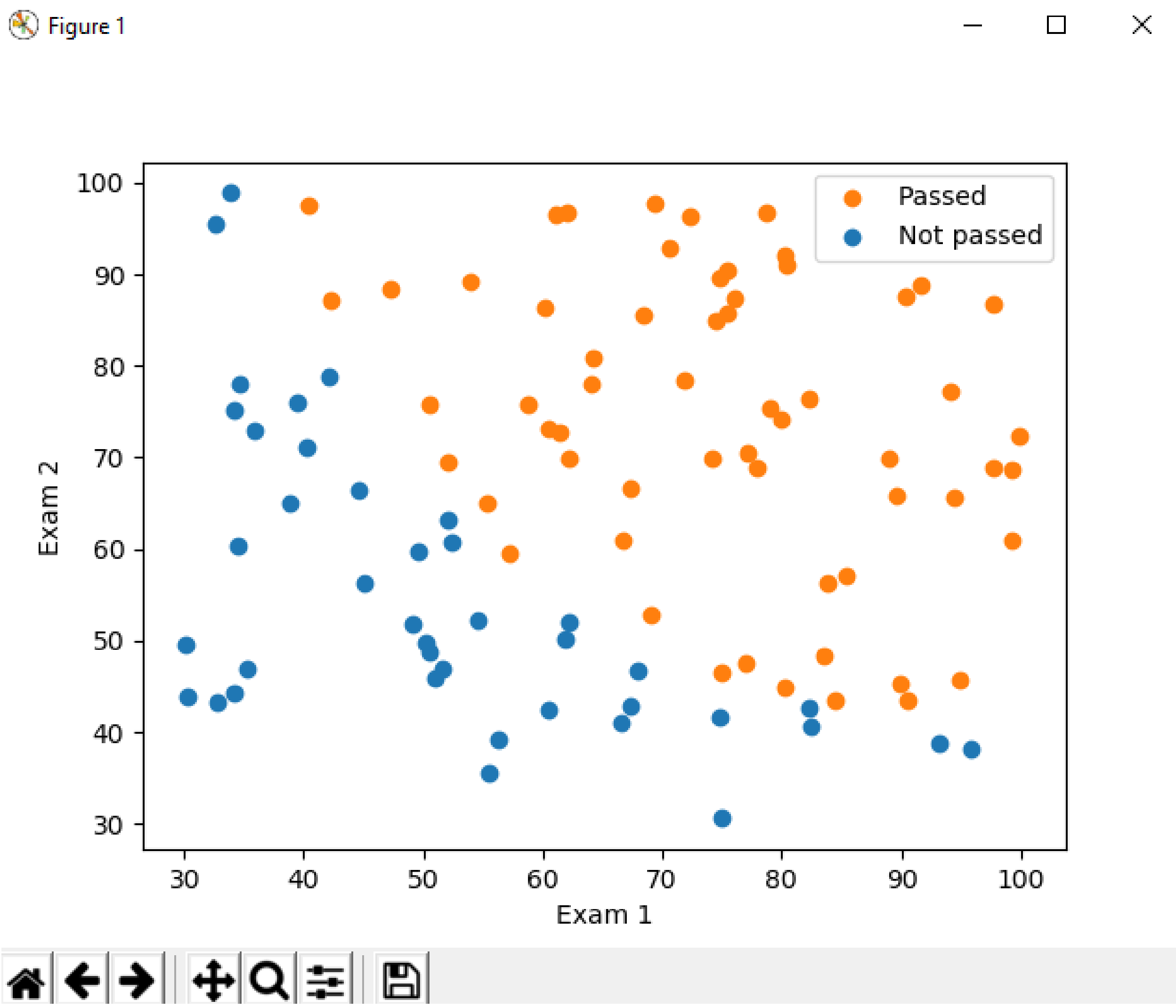
Минск 2019

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

```
#1
data = pd.read_csv('ex2data1.txt', header=None)
num_columns = data.shape[1]
X = data.iloc[:, 0:num_columns - 1]
y = data[num_columns - 1]
```

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.

```
false_indexes = y == 0
true_indexes = y == 1
fail = plt.scatter(X[false_indexes][0].values, X[false_indexes][1].values)
ok = plt.scatter(X[true_indexes][0].values, X[true_indexes][1].values)
plt.xlabel('Exam 1')
plt.ylabel('Exam 2')
plt.legend((ok, fail), ('Passed', 'Not passed'))
plt.show()
```



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

```
def loss(w, X, y):
    h = sig(np.dot(X, w))
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

def gradient_descent(X, y, w, learning_rate=0.0001, k=0.005, steps=105000):
    t = 1
    next_w = w - k * gradient(w, X, y)
    while np.linalg.norm(w - next_w) > learning_rate and t < steps:
        w = next_w
        next_w = w - k * gradient(w, X, y)
        t += 1
    return next_w

def gradient(w, X, y):
    h = sig(np.dot(X, w))
    return np.dot(X.T, (h - y)) / len(y)
```

$$\theta_{optimal} = [-25.16130062 \quad 0.20623142 \quad 0.20147143]$$
$$F_{loss} = 0.2034$$

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).

```
temp = optimize.minimize(loss, np.array([0, 0, 0]), (X, y), method='Nelder-
Mead')
print(temp.x)
print(loss(temp.x, X, y))
theta_optimized = optimize.fmin_bfgs(
    loss,
    np.array([0, 0, 0]),
    gradient,
    (X, y)
)
print(theta_optimized)
print(loss(theta_optimized, X, y))
```

Метод Нелдера — Мида:

$$\theta_{optimal} = [-25.16130062 \quad 0.20623142 \quad 0.20147143]$$
$$F_{loss} = 0.2034$$

Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно:

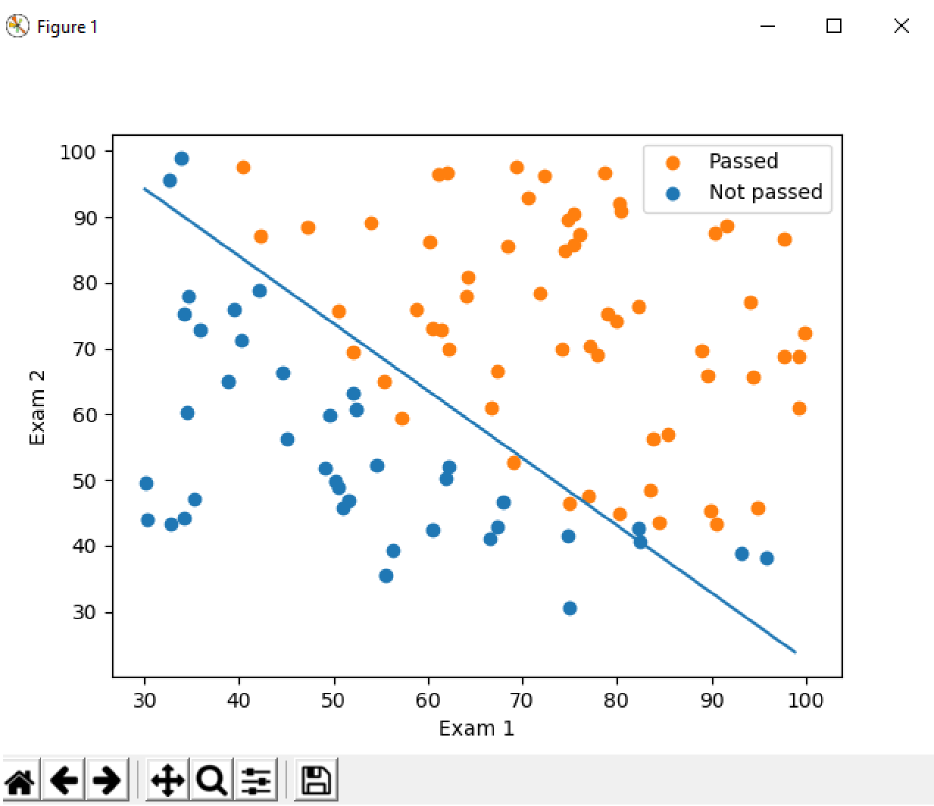
$$\theta_{optimal} = [-25.16130062 \quad 0.20623142 \quad 0.20147143]$$
$$F_{loss} = 0.2034$$

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

```
def predict_student_exam(X, theta):
    h = sig(np.dot(X.T, theta))
    return h >=0.5
```

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

```
x1 = np.array([np.min(X[0]), np.max(X[1])])
x2 = (-1 / theta_optimized[2]) * (x1 * theta_optimized[1]) + offset
plt.plot(x1, x2)
plt.show()
```

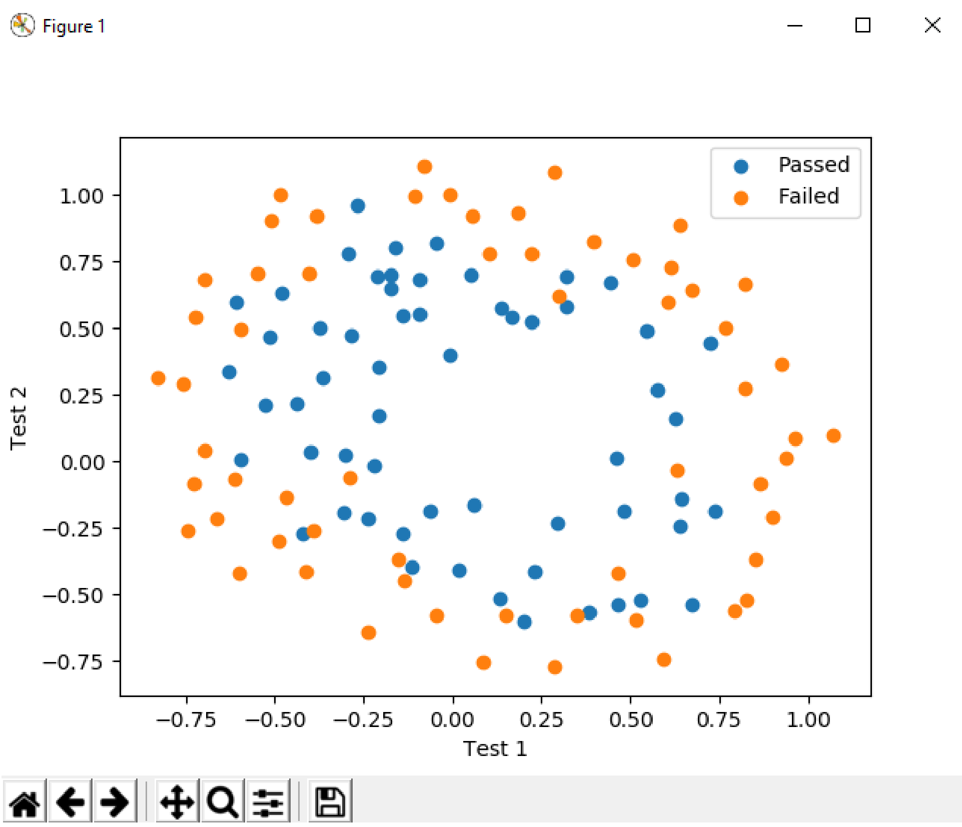


7) Загрузите данные **ex2data2.txt** из текстового файла.

```
data = pd.read_csv('ex2data2.txt', header=None)
num_columns = data.shape[1]
X = data.iloc[:, 0:num_columns - 1]
y = data[num_columns - 1]
```

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

```
ok = y == 1
fail = y != 1
psd = plt.scatter(X[ok][0].values, X[ok][1].values)
not_psd = plt.scatter(X[fail][0].values, X[fail][1].values)
plt.xlabel('Test 1')
plt.ylabel('Test 2')
plt.legend((psd, not_psd), ('Passed', 'Failed'))
plt.show()
```



9) Постройте все возможные комбинации признаков x_1 (результат первого теста) и x_2 (результат второго теста), в которых степень полинома не превышает 6, т.е. 1, x_1, x_2, x_1^2, x_1x_2, x_2^2, …, $x_1x_2^5$, x_2^6 (всего 28 комбинаций).

```
def polynom_combs(x1, x2):
    res = []
    for i in range(0,7):
        for j in range(0,7):
            if i + j <= 6:
                res.append((x1**i)*(x2**j))
    return  res
```

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

```
def gradient_regularized(theta, X, y, k):
    m = len(y)
    grad = gradient(theta, X ,y)
    grad[1:] = grad[1:]+ (k / m) * theta[1:]
    return grad

def loss_regularized(theta, X,y, k=0.0):
    h = sig(np.dot(X, theta))
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean() + (k / (2 * m))
* np.sum(theta[1:]**2)
```

$$\theta_{optimal}(\lambda = 0.1) = [\,2.75363564e+00\ \ 2.55125641e-04\ \ 2.95665612e+00\ \ -4.22534131e+00$$

$$-4.99263822e-01\ \ -2.77812164e+00\ \ -9.15659011e-02\ \ -1.14871499e+00$$

$$1.80692897e+00\ \ -3.37974660e+00\ \ -4.71996190e-01\ \ -1.20769720e+00$$

$$-1.21194951e+00\ \ -9.58055385e-01\ \ -4.21454682e+00\ \ -1.07771751e+00$$

$$-1.76331744e+00\ \ -1.13700071e+00\ \ -1.14911219e+00\ \ 7.46372415e-01$$

$$5.27887416e-01\ \ 6.23622967e-01\ \ 4.26553258e-01\ \ -3.26578450e+00$$

$$-4.70348678e-01\ \ -7.36392591e-01\ \ -6.21952129e-01\ \ 4.45546465e-01$$

$$-2.63113683e+00]$$

$$F_{loss} = 0.35237$$

11) Реализуйте другие методы оптимизации. https://github.com/Karpengold/machine_learning/blob/master/lab2/index.py#L159

$$\theta_{optimal}(\lambda = 0.1, Nelder - Mead\ algorithm) = [\,2.11170909\ \ 0.02964283\ \ 1.76836171\ -2.73423655\ -0.91189914\ -1.00303153$$

$$-0.57557263\ -0.94737048\ \ 0.69174322\ -2.14012523\ \ 0.70224413\ -2.97302144$$

$$1.20126705\ \ 1.14841242\ -5.44985121\ \ 0.31731831\ -1.40403439\ \ 1.37292496$$

$$1.54884457\ -0.04714634\ \ 3.85616341\ -2.34072884\ \ 0.67137147\ -0.97055022$$

$$1.9997212\ \ 2.93764183\ \ 2.50172441\ -1.88048827\ -1.84915092]$$

$$F_{loss} = 0.43$$

$$\theta_{optimal}(\lambda = 0.1, Brovden\ Fletcher\ Goldfarb\ Shanno\ algorithm) = [\,2.75328185e+00\ \ 3.09669282e-04\ \ 2.95834870e+00\ -4.22430586e+00$$
$$-5.01101147e-01\ -2.77877501e+00\ -9.24782096e-02\ -1.14755524e+00$$
$$1.80645753e+00\ -3.37821489e+00\ -4.71388600e-01\ -1.20752744e+00$$
$$-1.21285226e+00\ -9.58662463e-01\ -4.21322484e+00\ -1.08263255e+00$$
$$-1.76251166e+00\ -1.13840560e+00\ -1.14960135e+00\ \ 7.45659815e-01$$
$$5.27166460e-01\ \ 6.25286997e-01\ \ 4.27629808e-01\ -3.26591561e+00$$
$$-4.73787747e-01\ -7.35997854e-01\ -6.22082557e-01\ \ 4.43658768e-01$$
$$-2.63211063e+00]$$

$$F_{loss} = 0.35237$$

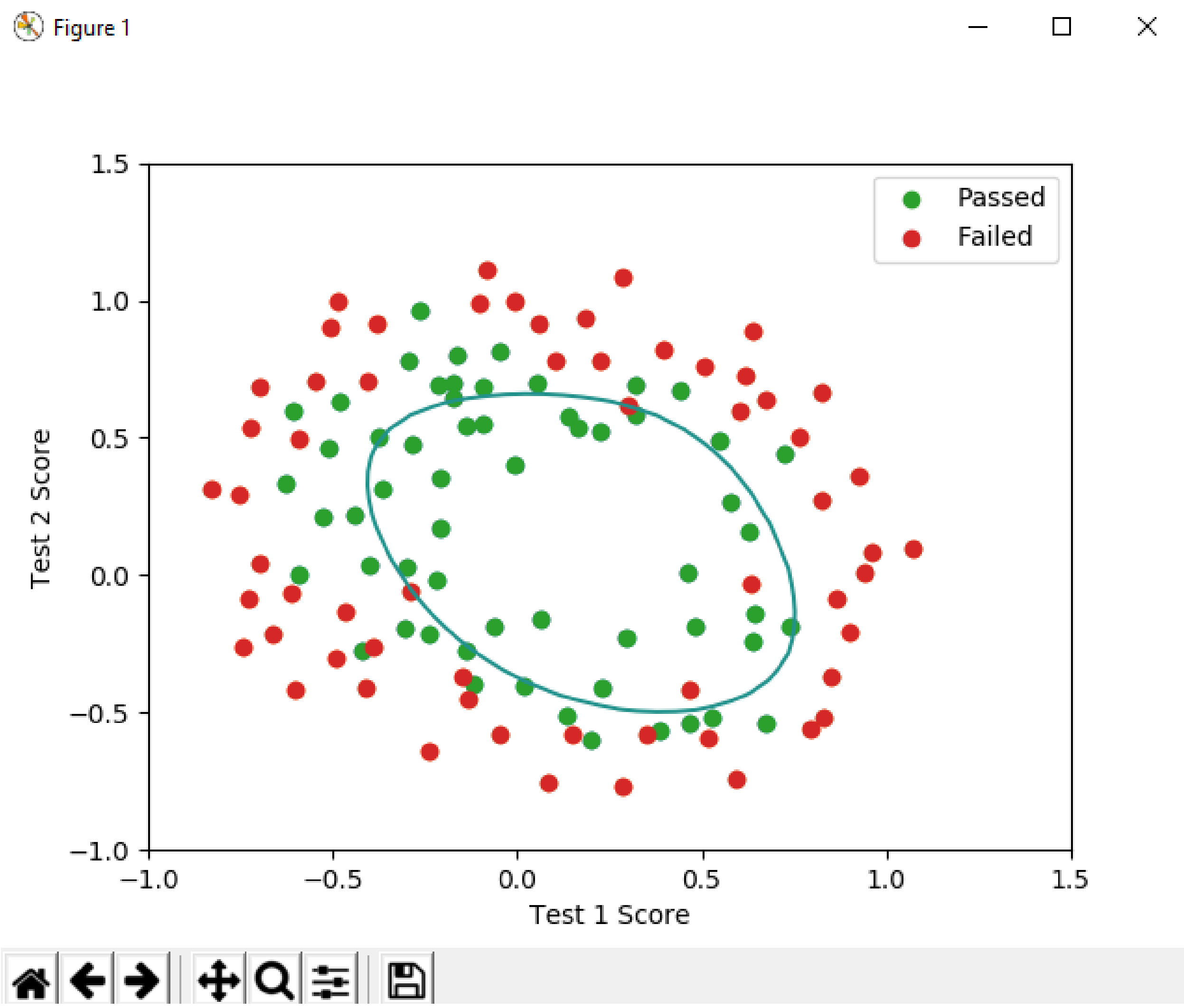
12) Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.

```
#12
print('Predictions: ', sig(np.dot(X.values[0].T, theta_bfgs)))
```

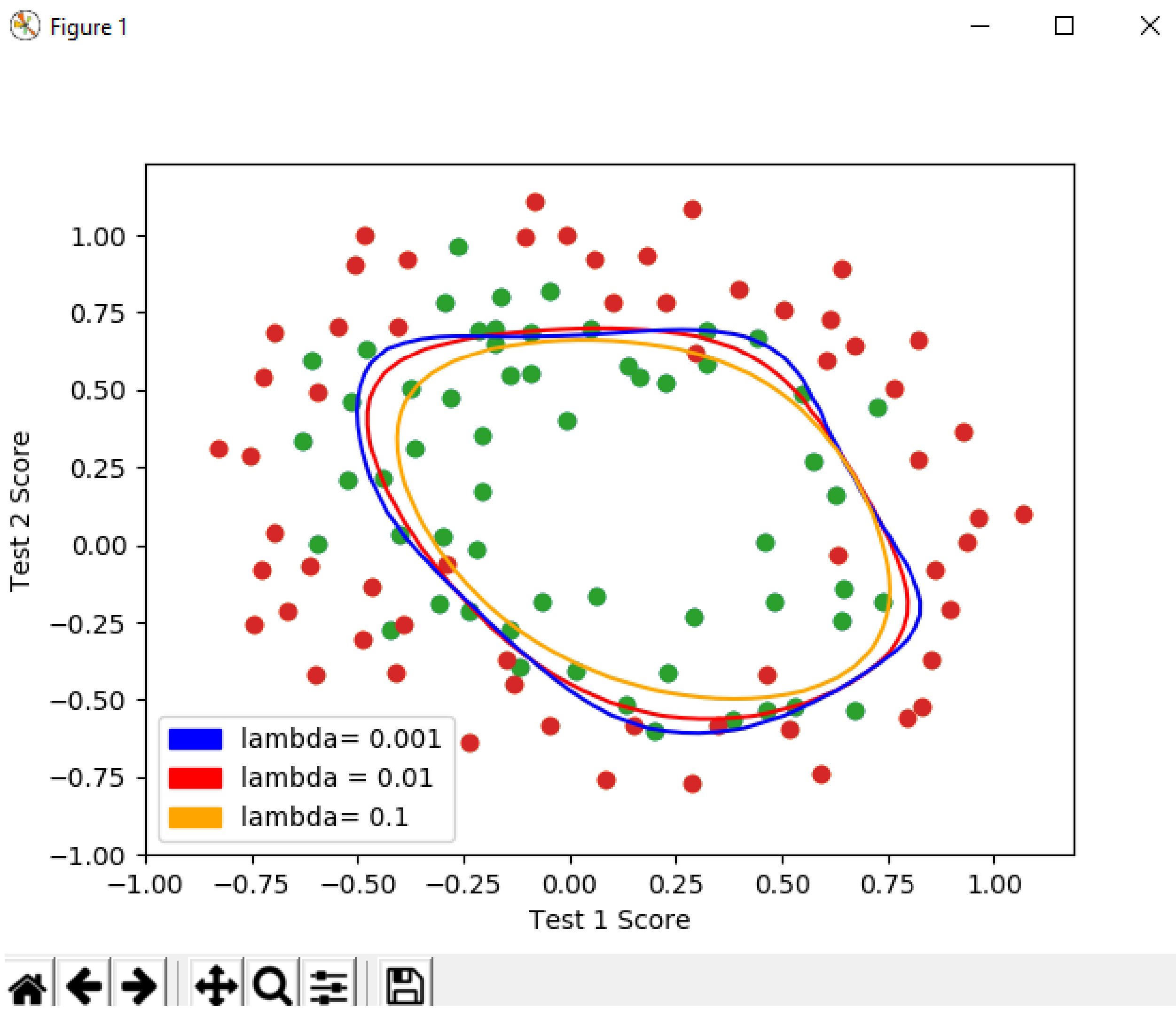
13)Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.

```
#13-14
theta_bfgs2 = optimize.fmin_bfgs(
    loss_regularized,
    theta.flatten(),
    gradient_regularized,
    (X, y, 0.001)
)
theta_bfgs3 = optimize.fmin_bfgs(
    loss_regularized,
    theta.flatten(),
    gradient_regularized,
    (X, y, 0.1)
)
xd = np.linspace(-1, 1, 50)
yd = np.linspace(-1, 1, 50)
z1 = np.zeros((len(xd), len(yd)))
z2 = np.zeros((len(xd), len(yd)))
z3 = np.zeros((len(xd), len(yd)))
for i in range(len(xd)):
    for j in range(len(yd)):
        dots = [1]
        dots.extend(polynom_combs(xd[i], yd[j]))
        z1[i, j] = sig(np.dot(np.array(dots).T, theta_bfgs))
for i in range(len(xd)):
    for j in range(len(yd)):
        dots = [1]
        dots.extend(polynom_combs(xd[i], yd[j]))
        z2[i, j] = sig(np.dot(np.array(dots).T, theta_bfgs2))
for i in range(len(xd)):
    for j in range(len(yd)):
        dots = [1]
        dots.extend(polynom_combs(xd[i], yd[j]))
        z3[i, j] = sig(np.dot(np.array(dots).T, theta_bfgs3))

mask = y.values.flatten() == 1
X = data.iloc[:, :-1]
passed = plt.scatter(X[mask][0], X[mask][1])
failed = plt.scatter(X[~mask][0], X[~mask][1])
plt.contour(xd, yd, z1, 0, colors='red')
plt.contour(xd, yd, z2, 0, colors= 'blue')
plt.contour(xd, yd, z3, 0, colors= 'orange')
blue_patch = mpatches.Patch(color='blue', label='lambda= 0.001')
red_patch = mpatches.Patch(color='red', label='lambda = 0.01')
orange_patch = mpatches.Patch(color='orange', label='lambda= 0.1')
plt.legend(handles=[blue_patch, red_patch, orange_patch])

plt.xlabel('Test 1 Score')
plt.ylabel('Test 2 Score')
```

```
plt.show()
```



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

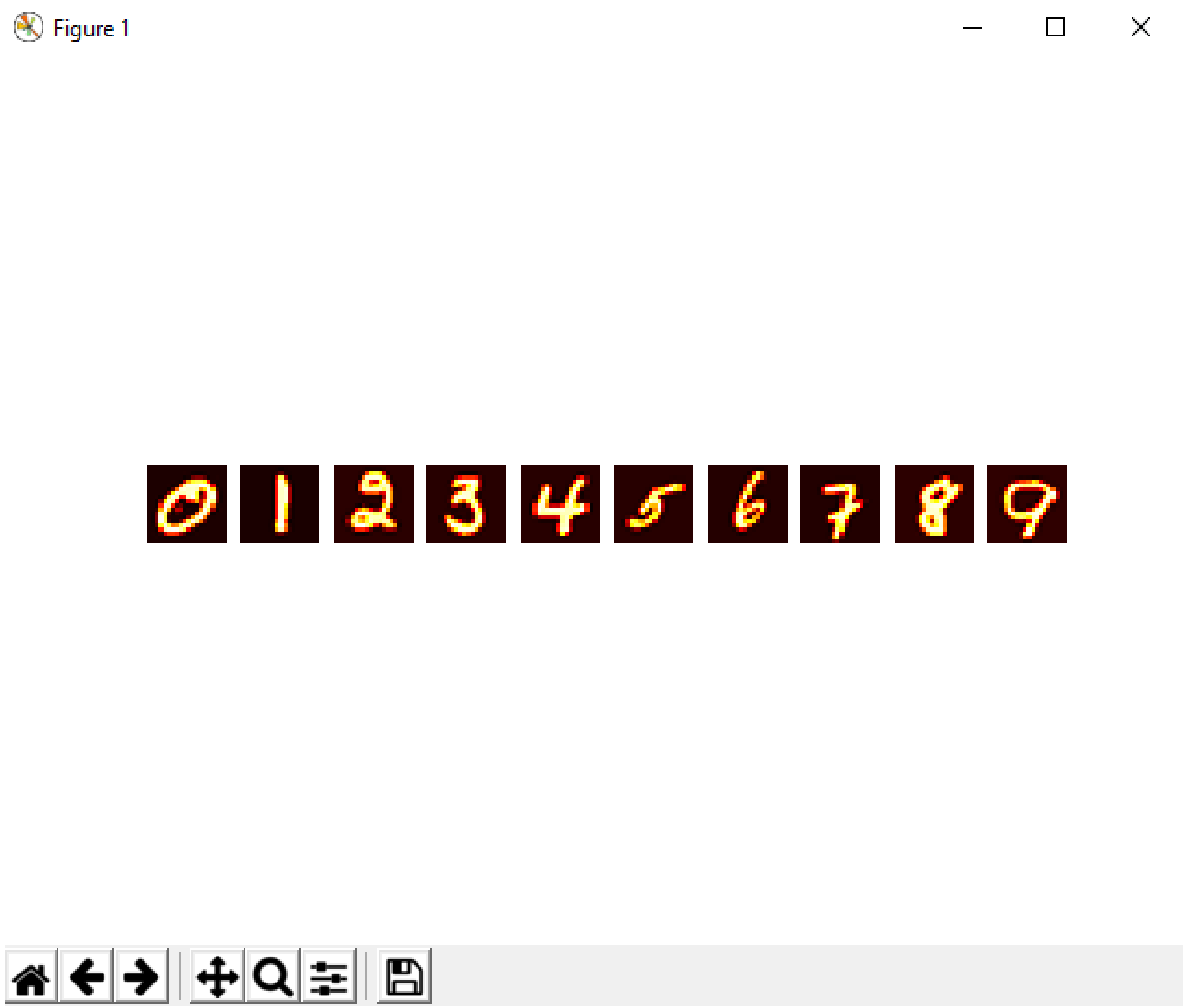


15) Загрузите данные **ex2data3.mat** из файла.

```
data = sio.loadmat('ex2data3.mat')
X = data.get('X')
y = data.get('y')
```

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

```
images = {}

for i in range(len(y)):
    images[y[i][0]] = i
keys = images.keys()

fig, axis = plt.subplots(1, 10)

for j in range(len(keys)):
    axis[j].imshow(X[images.get(list(keys)[j]), :].reshape(20, 20,
order="F"), cmap="hot")
    axis[j].axis("off")
```



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

```
#17
X = pd.DataFrame(X)
X.insert(0, 'Ones', 1)

(m, n) = X.shape
lmbda = 0.1
k = 10
theta = np.zeros((n, 1))  # initial parameters
print("F_loss ", loss(theta, X, y))
print("Gradient F_loss ", gradient(theta, X, y))
```

$$\theta = \bar{0}$$

$$F_{loss} = -17.0514$$

18) Добавьте L2-регуляризацию к модели.

```
#18
print("Loss regularized ", loss_regularized(theta, X, y, 0.01))
print("Gradient regularized F_loss ", gradient_regularized(theta, X, y,
0.01))
```

$$\lambda = 0.01$$
$$\theta = \bar{0}$$
$$F_{loss} = -17.0514$$

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

```
#19
theta_arr = []*10

for i in range (0, 10):
    print(i)
    digit_class = i if i else 10
    theta_temp = optimize.fmin_bfgs(
        loss_regularized,
        theta.flatten(),
        gradient_regularized,
        (X, (y == digit_class).flatten().astype(np.int), 0.1)
    )
    theta_arr.append(theta_temp)
```

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

```
def predict_number(theta_arr, x):
    return np.dot(x, np.array(theta_arr).T)

print('predict: ',  np.argmax(predict_number(X.values[0], theta_arr)), "Real:
", y[0][0])
```

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

```
#21
success = 0
predicted = predict_number(X.values, theta_arr).T
for i in range(len(predicted)):
    p = np.argmax(predicted[i])
    if  p == 0:
        p = 10
    if  p == y[i][0]:
        success += 1
print('Acc: ', success/len(predicted))
```

$$Accuracy = 0.9648$$

Выводы

В рамках лабораторной работы была изучена логистическая регрессия с использованием градиентного спуска в качестве метода минимизации функцию потерь, а также метод регуляризации.

Так как логистическая регрессия также строит линейную модель, как и линейная регрессия, она включает в себя те же минусы и плюсы:

Плюсы:

- Хорошо изучены
- Очень быстрые, могут работать на очень больших выборках
- Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные
- Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

Минусы:

- Плохо работают в задачах, в которых зависимость ответов от признаков сложная, нелинейная

Помимо этого можно сказать, что логистическая регрессия в задачах классификации выдает вероятности отнесения к разным классам (это очень ценится, например, в кредитном скоринге).