Лабораторная работа №2 “Логистическая регрессия. Многоклассовая классификация”

Долматович Алина, 858641

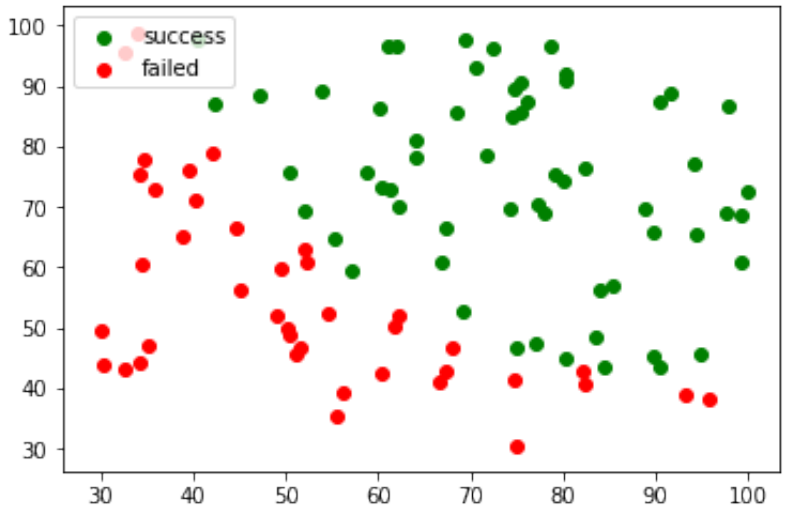
In [1]:
```
import pandas
import matplotlib.pyplot as pyplot
import numpy as np
import math
import scipy.optimize as optimize
from scipy.io import loadmat
import random
```

Загрузите данные ex2data1.txt из текстового файла.

In [2]:
```
def getData(fileName):
    return pandas.read_csv(fileName, sep = ",", header=None)

data1 = getData('ex2data1.txt')
# print(data1)
```

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 [3]:
```
def scatterPlot(data):
    colors = ("green", "red")
    groups = ("success", "failed")

    figure = pyplot.figure()
    subplot = figure.add_subplot(1, 1, 1)

    for data, color, group in zip(data, colors, groups):
        x, y = data[0], data[1]
        subplot.scatter(x, y, c=color, label=group)

    pyplot.legend(loc=2)

success = data1.loc[data1[2] == 1]
failed = data1.loc[data1[2] == 0]
scatterPlot(data=(success, failed))
pyplot.show()
```



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

In [12]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def h(theta, X):
    z = np.dot(theta, X.T)
    return sigmoid(z)

def particularCost(theta, x, y):
    if y == 1:
        return - math.log(h(theta, x))
    elif y == 0:
        return - math.log(1 - h(theta, x))

def cost(theta, X, Y):
    particularCosts = []
    for x, y in zip(X,Y):
        currentParticularCost = particularCost(theta, x, y)
        particularCosts.append(currentParticularCost)
    return 1. / len(X) * sum(particularCosts)

def gradientDescent(x, y, theta, alpha=0.0005):
    for i in range(500):
        h = sigmoid(np.dot(x, theta))
        gradient = np.matmul(X.T, (h - y)) / len(y);
        theta -= alpha * gradient
    return theta
```

In [5]:
```
X = np.array(data1.iloc[:, 0:2])
Y = np.array(data1[2])
theta = np.array([0, 0.1, 0.1])

(m, n) = X.shape
# theta = np.zeros((n + 1, 1))
X = np.hstack((np.ones((m, 1)), X))

print(cost(theta, X, Y))

theta = gradientDescent(X, Y, theta)
print(theta)
```

```
4.26613842362
[-0.01889155  0.01055634  0.00059828]
```

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).

In [6]:
```
temp = optimize.minimize(cost, theta, (X, Y), method='Nelder-Mead')
resultTheta = temp.x
print(resultTheta)

temp2 = optimize.minimize(cost, theta, (X, Y), method='BFGS')
resultTheta = temp2.x
print(resultTheta)
```

```
[-25.16142618   0.20623229   0.20147249]
[-25.16124435   0.206231     0.20147085]
```

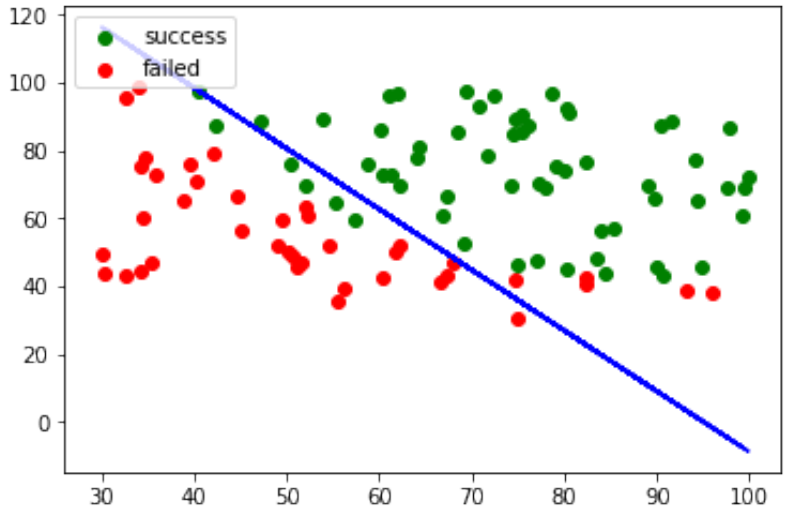
Реализуйте функцию предсказания вероятности поступления студента в зависимости от значений оценок по экзаменам.

In [7]:
```
h = h(resultTheta.copy(), X.copy())
print(h)
```

```
[  9.10342800e-02   4.22601706e-05   4.39091754e-02   9.90424545e-01
   9.98198714e-01   1.07976791e-02   9.98978114e-01   4.23227182e-01
   9.99710012e-01   7.35387352e-01   9.09673865e-01   2.85970699e-02
   9.99270125e-01   9.99853941e-01   1.56037429e-01   9.80959079e-01
   4.27716919e-01   1.48789818e-01   9.99706535e-01   5.74280445e-01
   6.68354987e-02   9.98627216e-01   7.41812959e-03   1.01158024e-04
   9.91878123e-01   8.55047779e-01   6.00298381e-01   8.65184223e-01
   9.23898049e-02   1.68290877e-02   8.97130010e-01   9.81037118e-01
   1.54911474e-01   3.91938487e-01   7.41293603e-02   3.31110712e-02
   8.52266888e-01   9.87564125e-01   2.03988405e-01   4.95802746e-02
   9.70300054e-01   6.11832483e-03   9.99461472e-01   5.02285115e-01
   4.48877376e-03   1.37024401e-01   9.92991691e-01   9.99996181e-01
   9.99204897e-01   9.99990760e-01   9.98106744e-01   9.99500476e-01
   9.05015341e-01   2.80619156e-03   8.52283234e-03   5.28913438e-02
   9.99856626e-01   6.93706428e-01   9.85496467e-01   9.95728534e-01
   9.99531636e-01   2.22382092e-04   3.50546294e-03   1.27070078e-04
   7.16571125e-02   4.08764977e-02   9.44425563e-01   1.00706754e-02
   9.99952446e-01   7.09299878e-01   6.20412637e-05   9.77395873e-01
   9.99893355e-01   8.84275600e-01   9.05239847e-01   9.99954903e-01
   9.17694064e-01   6.26742965e-01   1.58403269e-02   5.99468870e-01
   9.99282476e-01   9.73471313e-01   8.94531428e-01   2.03193080e-01
   9.99941046e-01   9.97982437e-01   3.54559867e-01   9.99820024e-01
   9.99973210e-01   1.06959730e-01   9.99943951e-01   9.99985224e-01
   1.42430938e-03   9.99321725e-01   9.24569617e-01   8.58638371e-01
   7.50880784e-01   9.99896603e-01   3.39275511e-01   9.99750926e-01
]
```

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

In [8]:
```
# -(t1*x1 + t3) / t2 = x2

def line_x1(x1, theta):
    result = []
    for x in x1:
        value = (theta[0] * x + theta[2]) / theta[1]
        result.append(value)
    return result

scatterPlot(data=(success, failed))
pyplot.plot(data1[0], line_x1(data1[0], theta), color='blue')
pyplot.show()
```

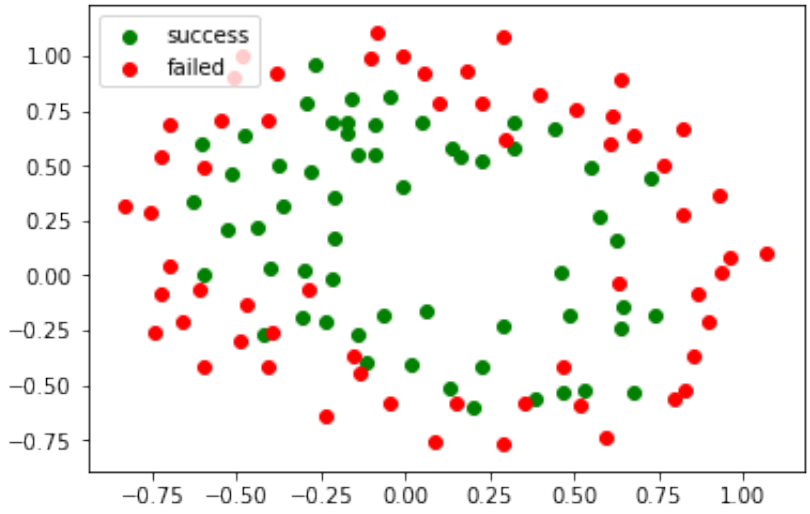


Загрузите данные ex2data2.txt из текстового файла.

In [9]:
```
data2 = getData('ex2data2.txt')
# print(data2)
```

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 [10]:
```
success = data2.loc[data2[2] == 1]
failed = data2.loc[data2[2] == 0]

scatterPlot(data=(success, failed))
pyplot.show()
```



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

In [13]:
```
#J(X,Y,W) = L + 1/C * ||W||^2, где ||W||^2 = w1^2 + w2^2 +...+wn^2, L 

X = np.array(data2.iloc[:, 0:2])
(m, n) = X.shape
theta = np.zeros((n + 1, 1))
X = np.hstack((np.ones((m, 1)), X))

Y = np.array(data2[2])
theta = np.array([0, 0.1, 0.2])

def l2regularization(theta, X, Y, C=0.1):
    particularCosts = []
    for x, y in zip(X,Y):
        currentParticularCost = particularCost(theta, x, y)  + (C / (2
        particularCosts.append(currentParticularCost)
    return 1. / len(X) * sum(particularCosts)

theta2 = gradientDescent(X, Y, theta)
L2 = l2regularization(theta, X, Y)
print(L2, theta2)
```

```
(0.69647643833990047, array([-0.00456763,  0.09412745,  0.19643552])
)
```

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

In [14]:
```
def partialPredict(h, y):
    return sigmoid(np.dot(h, y))

def prediction(X, Y):
    H = h(resultTheta.copy(), X)

    P = []
    for _h, y in zip(H, Y):
        partial = partialPredict(_h, y)
        P.append(partial)
    return P

predict = prediction(X, Y)
print(len(predict))
```

```
118
```

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, x12, x1x2, x22, …, x1x25, x26 (всего 28 комбинаций).

In [15]:
```
X = np.array(data2.iloc[:, 0:2])
Y = np.array(data2[2])
(m, n) = X.shape
theta = np.zeros((n + 1, 1))

def polynomialCombinations():
    def partialCombination(p1, p2):
        def multiply(x):
            return (x[0] ** p1) * (x[1] ** p2)

        return ['x1%s*x2%s' % (p1, p2), multiply]

    combinations = {}
    for i in range(0, 7):
        for j in range(0, 7):
            if i + j <= 6:
                [key, fn] = partialCombination(i, j)
                combinations[key] = fn
    return combinations

XX = []
for i in X:
    a = []
    combinations = polynomialCombinations()
    for key in combinations.keys():
        a.append(combinations[key](i))
    XX.append(np.array(a))
X = np.array(XX)
print(X.shape)
```

```
(118, 28)
```

Реализуйте другие методы оптимизации.

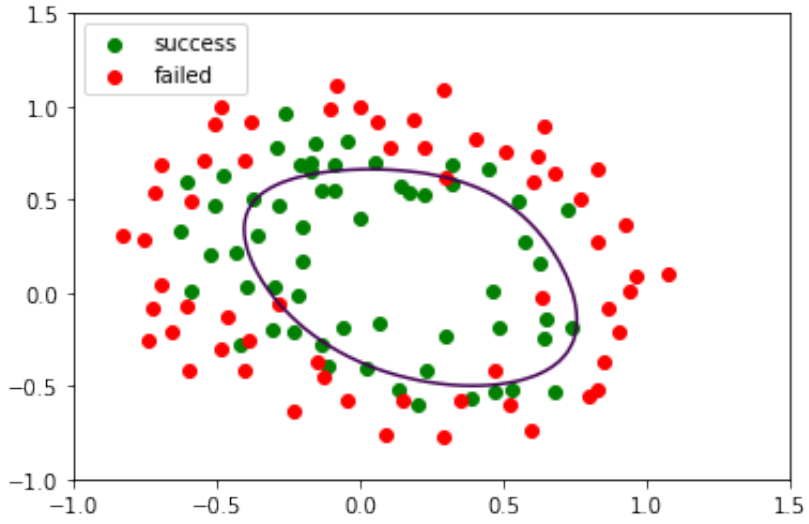
In [16]:
```
def getThetaOptimized(X, C=0.01):
    (m, n) = X.shape
    theta = np.zeros((n + 1, 1))
    X = np.hstack((np.ones((m, 1)), X))

    theta_optimized = optimize.minimize(l2regularization, theta, (X, Y
    print(theta_optimized)

#     theta_optimized = optimize.minimize(l2regularization, theta, (X,
#     print(theta_optimized)

    return theta_optimized

thetaOptimize = getThetaOptimized(X)
```

```
[ 3.73959371  3.23362165  4.64217887  2.97725137 -2.53605451 -6.3006
0183
  2.84732584  0.32110738  0.04666409  5.01139142 -6.05869898  2.1376
0524
 -3.6594982  -4.32656123 -4.49457957 -3.70160141 -3.60947846 -5.5621
0576
 -0.09605167  0.41744376 -1.70520988 -5.48256526 -6.69650402  2.0267
6681
 -1.98244411 -3.75115211 -0.87408806 -0.85685924  2.49057132]
```

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

In [17]:
```
def dividingCurve(C):
    thetaOptimized = getThetaOptimized(X, C)
    XX = np.linspace(-1, 1.5, 100)
    YY = np.linspace(-1, 1.5, 100)
    ZZ = np.zeros((len(XX), len(YY)))
    combinations = polynomialCombinations()

    for i in range(len(XX)):
        for j in range(len(YY)):
            a = [1]
            for key in combinations.keys():
                a.append(combinations[key]([XX[i], YY[j]]))
            ZZ[i, j] = h(thetaOptimized, np.array(a))

    success = data2.loc[data2[2] == 1]
    failed = data2.loc[data2[2] == 0]
    scatterPlot(data=(success, failed))
    pyplot.contour(XX, YY, ZZ, 0)

dividingCurve(0.1)
pyplot.show()
```

```
[  2.75511335e+00   4.27654981e-01   2.95675339e+00   5.27832419e-01
  -4.99821618e-01  -2.77958684e+00  -9.17935165e-02  -1.14888962e+00
  -1.22050153e-03   6.24093385e-01  -4.22406247e+00   7.46224940e-01
  -9.58722753e-01  -1.21465841e+00  -1.14921411e+00  -1.76212068e+00
  -1.13637721e+00  -4.21467546e+00  -1.07812657e+00   4.44274096e-01
  -6.21338533e-01  -2.63112452e+00  -3.37771914e+00   1.80722305e+00
  -1.20754102e+00  -3.26537172e+00  -4.71452579e-01  -7.35762809e-01
  -4.74778418e-01]
```

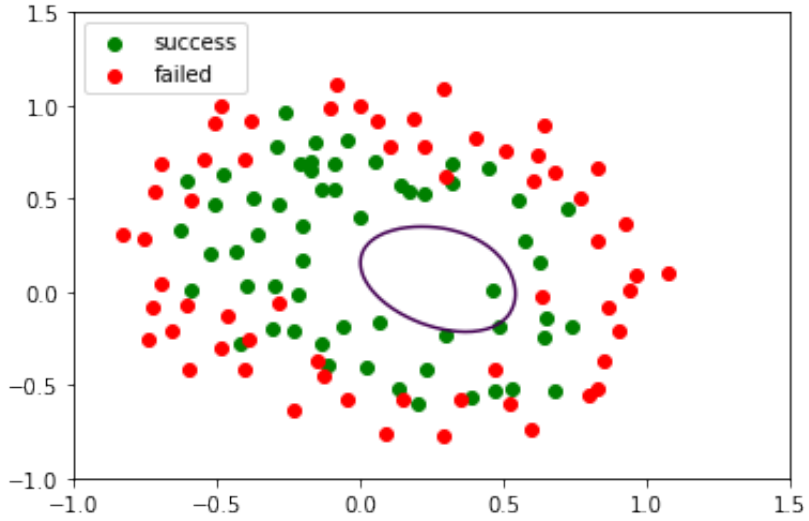


Попробуйте различные значения параметра регуляризации λ. Как выбор данного значения влияет на вид разделяющей кривой? Ответ дайте в виде графиков.

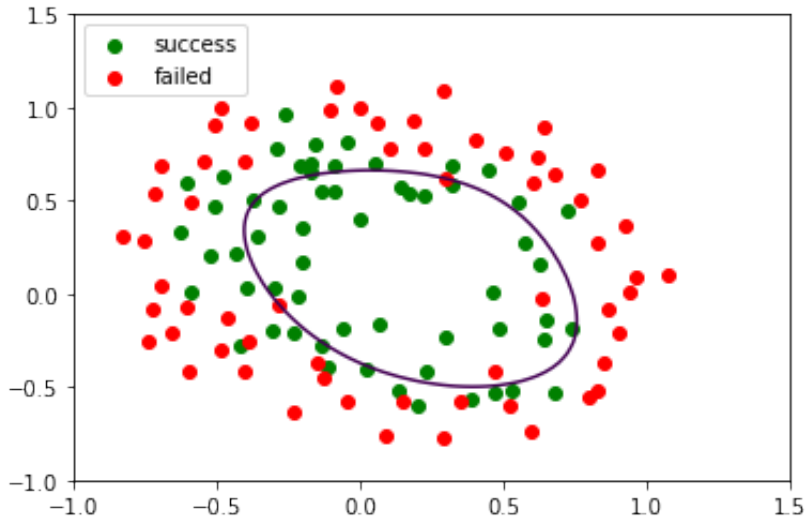
In [18]:
```
cValues = [0.9, 0.1, 0.01, 0.001]

for value in cValues:
```

```
    dividingCurve(value)
    pyplot.show()
```

```
[  1.33477867e+00   1.94091487e-02   1.25196521e+00  -4.95881899e-02
  -1.77273238e-01  -1.25575922e+00  -4.62986339e-01  -9.59079703e-01
  -1.77360113e-04  -4.11317435e-02  -1.52540465e+00   1.44419801e-01
  -1.59744279e-01  -3.19536273e-01  -3.48482111e-01  -6.53891858e-01
  -2.98062103e-01  -2.12258443e+00  -3.93026207e-01   3.37540950e-02
  -2.48913912e-01  -1.09682006e+00  -9.94359359e-01   6.74738528e-01
  -3.00536003e-01  -1.53257646e+00  -2.18956428e-01  -3.09485552e-01
  -3.79513258e-01]
```

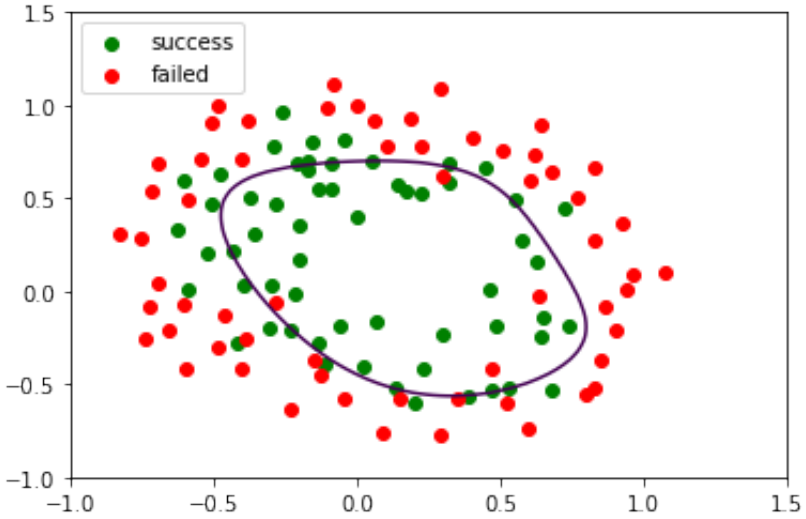



```
[  2.75511335e+00   4.27654981e-01   2.95675339e+00   5.27832419e-01
  -4.99821618e-01  -2.77958684e+00  -9.17935165e-02  -1.14888962e+00
  -1.22050153e-03   6.24093385e-01  -4.22406247e+00   7.46224940e-01
  -9.58722753e-01  -1.21465841e+00  -1.14921411e+00  -1.76212068e+00
  -1.13637721e+00  -4.21467546e+00  -1.07812657e+00   4.44274096e-01
  -6.21338533e-01  -2.63112452e+00  -3.37771914e+00   1.80722305e+00
  -1.20754102e+00  -3.26537172e+00  -4.71452579e-01  -7.35762809e-01
  -4.74778418e-01]
```

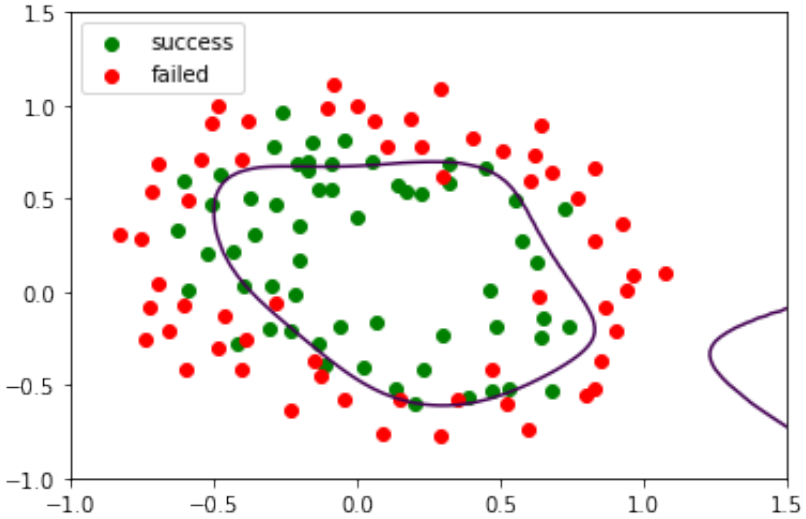



```
[ 3.73959371  3.23362165  4.64217887  2.97725137 -2.53605451 -6.3006
0183
```

```
  2.84732584  0.32110738  0.04666409  5.01139142 -6.05869898  2.1376
0524
 -3.6594982  -4.32656123 -4.49457957 -3.70160141 -3.60947846 -5.5621
0576
 -0.09605167  0.41744376 -1.70520988 -5.48256526 -6.69650402  2.0267
6681
 -1.98244411 -3.75115211 -0.87408806 -0.85685924  2.49057132]
```




```
[  3.0049419   11.76737158   4.64860032   3.18674038  -3.30114775
 -18.22425516   8.88881534   1.40932792   0.47215167  10.57478771
  -0.86041362   5.02722798 -13.34155455 -13.29903408 -17.39451255
 -10.11155637  -8.58889272  -5.61364754   6.80390598  -7.79178492
  -1.91957033 -18.03966713  -8.57339407   0.57987762  -1.00980683
   2.52891058  -3.74723557  -2.69935809  11.35061294]
```



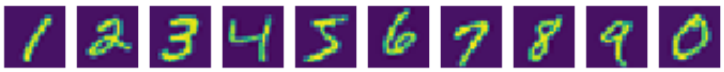
Загрузите данные ex2data3.mat из файла.

In [19]:
```
mat = loadmat('ex2data3.mat')
X = mat['X']
Y = mat['y']
print(X.shape)
print(Y.shape)
```

```
(5000, 400)
(5000, 1)
```

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

In [20]:
```
images = dict()

for i in range(len(Y)):
    key = Y[i][0]
    if key not in images.keys():
        images[key] = i

keys = images.keys()

_, axis = pyplot.subplots(1, len(keys))

for j in range(len(keys)):
    matrix = X[images[j+1], :].reshape(20, 20, order="F")
    axis[j].imshow(matrix)
    axis[j].axis("off")

pyplot.show()
```



In [21]:
```
m = len(Y)
X = np.hstack((np.ones((m, 1)), X))
(m, n) = X.shape
lmbda = 0.1
k = 10
theta = np.zeros((k, n))
```

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

In [22]:
```
def costRegularized(theta, X, y, lambda_=0):
    m = len(y)
    h_theta = sigmoid(np.dot(X, theta))
    J = (1. / m) * ((np.dot(-y.T, np.log(h_theta))) - np.dot((1 - y).T
    return J

def gradientDescentRegularized(theta, X, Y, alpha=0.01, lambdaValue=0)
    m = len(Y)
    grad = np.zeros([m, 1])
    grad = (1. / m) * np.dot(X.T, (sigmoid(np.dot(X, theta.T)) - Y))
    grad = grad.T + ((lambdaValue / m) * theta)
    return grad

print(costRegularized(theta[0], X, Y))
print(gradientDescentRegularized(theta, X, Y))
```

```
[-17.05142064]
[[ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 ...,
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]]
```

Добавьте L2-регуляризацию к модели.

In [23]:
```
lmbda = 0.01
print(costRegularized(theta[0], X, Y, lmbda))
print(gradientDescentRegularized(theta[0], X, Y, lmbda))
```

```
[-17.05142064]
[[ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 ...,
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]
 [ -5.00000000e+00   0.00000000e+00   0.00000000e+00 ...,   7.287646
24e-06
    6.02839052e-07   0.00000000e+00]]
```

Реализуйте многоклассовую классификацию по методу “один против всех”.

In [28]:
```
def oneVSAll(theta, k, X, Y, C=0.1):
    for i in range(k):
        digit_class = i if i else 10
        currentY = (Y == digit_class).flatten().astype(np.int)
        theta[i] = optimize.fmin_cg(
            f=costRegularized,
            x0=theta[i],
            fprime=gradientDescentRegularized,
            args=(X, currentY, C),
            maxiter=100
        )
    return theta

theta = oneVSAll(theta, k, X, Y)
print(theta)
```

```
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.008937
         Iterations: 63
         Function evaluations: 2608
         Gradient evaluations: 2597
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.013482
```

```
         Iterations: 18
         Function evaluations: 237
         Gradient evaluations: 225
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.051921
         Iterations: 6
         Function evaluations: 100
         Gradient evaluations: 88
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.058449
         Iterations: 38
         Function evaluations: 668
         Gradient evaluations: 656
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.034447
         Iterations: 8
         Function evaluations: 121
         Gradient evaluations: 109
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.055264
         Iterations: 9
         Function evaluations: 184
         Gradient evaluations: 172
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.020437
         Iterations: 9
         Function evaluations: 92
         Gradient evaluations: 81
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.031460
         Iterations: 16
         Function evaluations: 278
         Gradient evaluations: 266
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.079965
         Iterations: 100
         Function evaluations: 280
         Gradient evaluations: 280
Warning: Desired error not necessarily achieved due to precision los
s.
         Current function value: 0.071799
         Iterations: 37
         Function evaluations: 336
         Gradient evaluations: 324
[[ -7.41679593e+00   0.00000000e+00   0.00000000e+00 ...,  -4.125049
29e-04
    1.65380789e-05   0.00000000e+00]
```

```
 [ -3.19334873e+00   0.00000000e+00   0.00000000e+00 ...,   5.272751
14e-03
    2.59448167e-07   0.00000000e+00]
 [ -3.67211698e+00   0.00000000e+00   0.00000000e+00 ...,   1.003658
91e-02
   -1.14501572e-03   0.00000000e+00]
 ...,
 [ -2.23813374e+00   0.00000000e+00   0.00000000e+00 ...,  -1.897871
84e-03
    2.28927589e-04   0.00000000e+00]
 [ -9.26845069e+00   0.00000000e+00   0.00000000e+00 ...,  -2.457774
06e-04
    2.13078648e-05   0.00000000e+00]
 [ -5.72624956e+00   0.00000000e+00   0.00000000e+00 ...,  -6.963963
95e-03
    5.53371615e-04   0.00000000e+00]]
```

Реализуйте функцию предсказания класса по изображению с использованием обученных классификаторов.

In [25]:
```
def predict_number(X, theta):
    return np.argmax(np.dot(X, theta.T))

randomValue = int(random.uniform(1, len(X)))
print(randomValue)
print(predict_number(X[randomValue], theta), ' == ', Y[randomValue][0]
```

```
1797
(3, ' == ', 3)
```

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

In [29]:
```
predictions = 0
for index in range(len(X)):
    predict = 10 if predict_number(X[index], theta) == 0 else predict_
    real = Y[index][0]
    if predict == real:
        predictions += 1

print(predictions, len(Y))
print("Accuracy: ", (float(predictions) / len(Y)) * 100)
```

```
(4806, 5000)
('Accuracy: ', 96.12)
```