Лабораторная работа №2: «Логистическая регрессия. Многоклассовая классификация»

Набор данных ex2data1.txt представляет собой текстовый файл, содержащий информацию об оценке студента по первому экзамену (первое число в строке), оценке по второму экзамену (второе число в строке) и поступлении в университет (0 - не поступил, 1 - поступил).

Набор данных ex2data2.txt представляет собой текстовый файл, содержащий информацию о результате первого теста (первое число в строке) и результате второго теста (второе число в строке) изделий и результате прохождения контроля (0 - контроль не пройден, 1 - контроль пройден).

Набор данных ex2data3.mat представляет собой файл формата mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 20x20 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 400 элементов. В результате загрузки набора данных должна быть получена матрица 5000x400. Далее расположены метки классов изображений от 1 до 9 (соответствуют цифрам от 1 до 9), а также 10 (соответствует цифре 0).*¶

Задание:

Загрузите данные ex2data1.txt из текстового файла.

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.

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

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).

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

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

Загрузите данные ex2data2.txt из текстового файла.

1. Постройте график, где по осям откладываются результаты тестов, а точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет.
2. 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).
3. Реализуйте L2-регуляризацию для логистической регрессии и обучите ее на расширенном наборе признаков методом градиентного спуска.
4. Реализуйте другие методы оптимизации.
5. Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.
6. Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.
7. Попробуйте различные значения параметра регуляризации λ. Как выбор данного значения влияет на вид разделяющей кривой? Ответ дайте в виде графиков.
8. Загрузите данные ex2data3.mat из файла.
9. Визуализируйте несколько случайных изображений из набора данных. Визуализация должна содержать каждую цифру как минимум один раз.
10. Реализуйте бинарный классификатор с помощью логистической регрессии с использованием векторизации (функции потерь и градиентного спуска).
11. Добавьте L2-регуляризацию к модели.
12. Реализуйте многоклассовую классификацию по методу “один против всех”.
13. Реализуйте функцию предсказания класса по изображению с использованием обученных классификаторов.
14. Процент правильных классификаций на обучающей выборке должен составлять около 95%.
15. Ответы на вопросы представьте в виде отчета.

Реализация:

In[1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pnd
```

1.1 Загрузите набор данных ex2data1.txt из текстового файла.

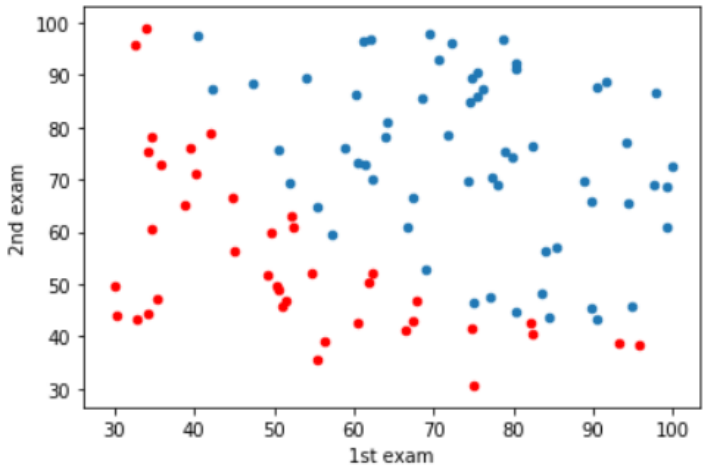
In[2]:

```
df = pd.read_csv('data/ex2data1.txt', header=None, names=['first_exam', 'sec-
ond_exam', 'accepted'])
x_train, y_train = df.filter(['first_exam', 'second_exam']), df['accepted']
```

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

In[3]:

```
df_accepted = df[df['accepted'] == 1]
df_not_accepted = df[df['accepted'] == 0]
fig, ax = plt.subplots()
ax.scatter(df_accepted['first_exam'], df_accepted['second_exam'], marker='o', la-
bel='Accepted', s=20)
ax.scatter(df_not_accepted['first_exam'], df_not_accepted['second_ex-
am'], marker='o', c='r', label='Not accepted', s=20)
ax.set_xlabel('1st exam')
ax.set_ylabel('2nd exam')
plt.show()
```



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

In[4]:

```
THRESHOLD = 1e-8

def sigmoid(z):
    return 1 / (1 + np.e ** (-z))

def h(x, theta):
    return sigmoid(x.dot(theta))

def cost_func_vectorized(x, y, theta, **kwargs):
    h_theta = h(x, theta)
    cost_1 = y * np.log(h_theta)
    cost_0 = (1 - y) * np.log(1 - h_theta)
    return -np.mean(cost_1 + cost_0)

def cost_func_deriative(x, y, theta, **kwargs):
    h_v = h(x, theta)
    gradient = np.dot(x.T, h(x, theta) - y)
    gradient /= x.shape[0]
    return gradient

def gradient_descent_vectorized(x, y, theta, a=1, max_iter_count=30000,
                                cost_func=cost_func_vector-
ized, cost_func_der=cost_func_deriative,
                                **kwargs):
    logs = []
    last_loss = cost_func(x, y, theta)
    for iter_num in range(max_iter_count):
        gradient = cost_func_der(x, y, theta, **kwargs)
        theta -= gradient * a
        curr_loss = cost_func(x, y, theta, **kwargs)

        logs.append([iter_num, curr_loss])

        if abs(curr_loss - last_loss) < THRESHOLD:
            break

        last_loss = curr_loss

    return theta, logs

def fit(x_train, y_train, minimization_func=gradient_descent_vectorized, regular-
ized=False,
        cost_func=cost_func_vectorized, cost_func_der=cost_func_deriat-
ive, **kwargs):

    x = getattr(x_train, 'values', x_train).astype('float64')
    y = getattr(y_train, 'values', y_train).astype('float64')

    if not regularized:
        x = np.column_stack((np.ones(x.shape[0]), x))
```

1.4 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).

In[5]:

```
    # Nelder-Mead method
def nelder_mead_algo(x, y, theta, cost_func=cost_func_vectorized, **kwargs):
    from scipy.optimize import fmin

    res_theta = fmin(lambda theta: cost_func(x, y, theta),
                     theta, xtol=THRESHOLD, maxfun=150000)
    return res_theta, []

# B-F-G-S method
def bfgs_algo(x, y, theta, cost_func=cost_func_vector-
ized, cost_func_der=cost_func_deriative, **kwargs):
    from scipy.optimize import fmin_bfgs

    res_theta = fmin_bfgs(lambda theta: cost_func(x, y, theta),
                          theta, fprime=lambda theta: cost_func_der(x, y, theta),
                          gtol=THRESHOLD)
    return res_theta, []

fit(x_train, y_train, minimization_func=bfgs_algo)
fit(x_train, y_train, minimization_func=nelder_mead_algo)
```

Out[5]:

```
Optimization terminated successfully.
         Current function value: 0.203498
         Iterations: 25
         Function evaluations: 33
         Gradient evaluations: 33
Optimization terminated successfully.
         Current function value: 0.203498
         Iterations: 198
         Function evaluations: 379


(array([-25.16133374,   0.20623172,   0.2014716 ]), [])
```

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

In[6]:

```
def predict(x, theta, regularized=False):
    x = np.array(x)
    if not regularized:
        x = np.insert(x, 0, 1)
    h_value = h(x, theta)
    return 1 if h_value >= 0.5 else 0
```

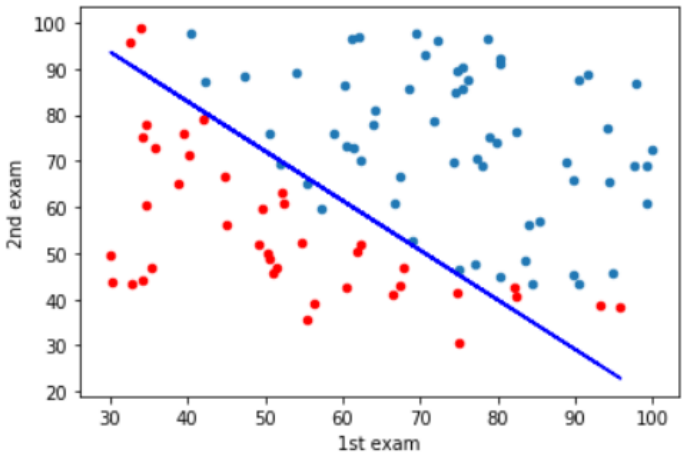
1.6. Постройте разделяющую прямую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 2.

In[7]:

```
theta, logs = fit(x_train, y_train)

def x2_func(x1, theta):
    return -(theta[0] + theta[1] * x1) / theta[2]

df_accepted = df[df['accepted'] == 1]
df_not_accepted = df[df['accepted'] == 0]
fig, ax = plt.subplots()
ax.scatter(df_accepted['first_exam'], df_accepted['second_exam'], marker='o', lab
el='Accepted', s=20)
ax.scatter(df_not_accepted['first_exam'], df_not_accepted['second_exam'], marker=
'o', c='r', label='Not accepted', s=20)
ax.plot(df_not_accepted['first_exam'],
        [x2_func(i, theta) for i in df_not_accepted['first_exam']],
        c='b', label='boundary')
ax.set_xlabel('1st exam')
ax.set_ylabel('2nd exam')
plt.show()
```




```
Кол-во итераций: 134116
```

1.7 Загрузите данные ex2data2.txt из текстового файла.

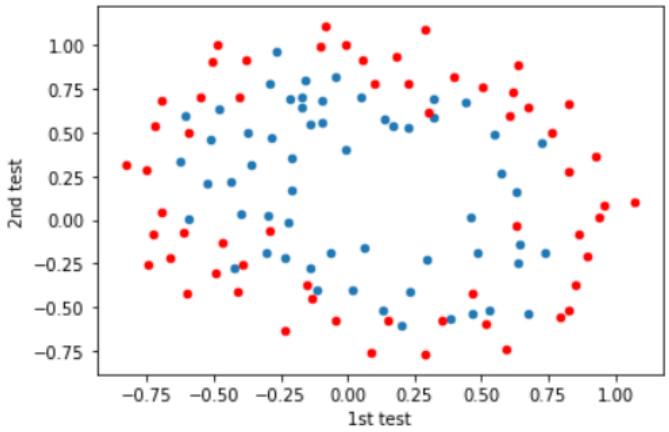
In[8]:

```
df = pd.read_csv('data/ex2data2.txt', header=None, names=['first_test', 'sec-
ond_test', 'passed'])
x_train, y_train = df.filter(['first_test', 'second_test']), df['passed']
df
```

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.¶

In[9]:

```
df_accepted = df[df['passed'] == 1]
df_not_accepted = df[df['passed'] == 0]
fig, ax = plt.subplots()
ax.scatter(df_accepted['first_test'], df_accepted['second_test'], marker='o', la-
bel='Passed', s=20)
ax.scatter(df_not_accepted['first_test'], df_not_accepted['sec-
ond_test'], marker='o', c='r', label='Not passed', s=20)
ax.set_xlabel('1st test')
ax.set_ylabel('2nd test')
plt.show()
```



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

In[10]

```
def build_features(x1, x2, degree):
    return [build_feature_for_pair(x1[idx], x2[idx], de-
gree) for idx in range(len(x1))]

extended_x_train = build_features(x_train['first_test'], x_train['sec-
ond_test'], 6)
extended_x_train = pd.DataFrame(extended_x_train)
```

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

In[11]:

```
def cost_func_vectorized_reg_l2(x, y, theta, penalty_term=0.1):
    cost = cost_func_vectorized(x, y, theta)
    theta_sliced = theta[1:]
    penalty_cost = (penalty_term / 2 / x.shape[0]) * np.dot(theta_sliced.T, theta
_sliced)
    return cost + penalty_cost

def cost_func_deriative_reg_l2(x, y, theta, penalty_term=0.1):
    err = h(x, theta) - y
    grad = np.dot(x.T[:1], err)
    grad_with_reg = np.dot(x.T[1:], err) + penalty_term * theta[1:]
    grad = np.insert(grad_with_reg, 0, grad)
    grad /= x.shape[0]
    return grad

theta, logs = fit(extended_x_train, y_train, minimization_func=gradient_descent_v
ectorized,
                  regularized=True, cost_func=cost_func_vectorized_reg_l2, cost_func
_der=cost_func_deriative_reg_l2)
theta, logs[-1][1]
```

Out[11]:

```
(array([ 2.47044991,  1.65382342, -3.94598303,  0.60846478, -3.03470089,
        -0.41870361, -2.27733632,  2.62767817, -2.69166142, -1.03075147,
        -0.00732896, -0.52230883,  0.15169121, -3.36472317, -0.83139245,
        -1.48613724,  0.07937035, -0.68211161, -0.22066692, -0.9332698 ,
        -0.78201023,  0.10385866, -2.50869643, -0.91520568, -0.83383276,
        -0.52838867, -0.60401967, -1.6019049 ]), 0.39766269795075404)
```

11. Реализуйте другие методы оптимизации.

In[12]:

```
theta1, logs1 = fit(extended_x_train, y_train, minimization_func=nelder_mead_algo
,
                    regularized=True, cost_func=cost_func_vectorized_reg_l2)
theta2, logs2 = fit(extended_x_train, y_train, minimization_func=bfgs_algo,
                    regularized=True, cost_func=cost_func_vectorized_reg_l2, cost_f
unc_der=cost_func_deriative_reg_l2)
theta1, theta2
```

Out[12]:

```
Optimization terminated successfully.
         Current function value: 0.465545
```

```
         Iterations: 27753
         Function evaluations: 33382
Optimization terminated successfully.
         Current function value: 0.394594
         Iterations: 136
         Function evaluations: 137
         Gradient evaluations: 137
```

Out[13]:

```
(array([ 2.3127117 ,  0.60394044, -5.58035455,  2.431011  , -1.79607751,
        -0.59007333, -0.52179195,  2.27876777, -1.82247495, -2.21304662,
         1.52207541,  2.68708933, -1.28528444, -2.74410273, -2.97284783,
         0.40094661,  2.10129466,  3.49785269, -1.78093335, -1.60371914,
         1.36959552,  0.60179941, -1.66845555,  2.15891957, -1.36062064,
        -0.08072131,  2.86706813,  0.04203006]),
 array([ 2.75388397,  1.80721584, -4.21449797,  0.74553063, -3.26575471,
        -0.62160039, -2.63126755,  2.95666289, -3.3795092 , -1.07820308,
         0.52791635, -0.47032487,  0.44537568, -4.22534326, -0.47234091,
        -1.76305288,  0.62375524, -0.73621705, -0.49912867, -1.20788555,
        -1.13701413,  0.42673693, -2.77794795, -1.21215545, -1.14910458,
        -0.09178338, -0.95793641, -1.14890432]))
```

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

Функция predict реализована выше. (Задание 5)