Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей

Специальность “Обработка больших объемов информации”

Лабораторная работа №2

**Логистическая регрессия. Многоклассовая классификация**

Выполнил:

магистрант гр. 858641 Кальман В.А.

Проверил:

Стержанов М. В.

Минск 2019

**Данные.**

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).

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

**from** **copy** **import** deepcopy

**from** **collections** **import** OrderedDict

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy** **import** io

**from** **scipy.optimize** **import** minimize

**from** **sklearn.preprocessing** **import** PolynomialFeatures

%matplotlib inline

ex2data1 = np.loadtxt('Data/Lab 2/ex2data1.txt', delimiter=',')

X, y = ex2data1[:, :2], ex2data1[:, 2]

**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.**

plt.figure(figsize=(16, 10))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

plt.xlabel('Mark 1', size=16)

plt.ylabel('Mark 2', size=16)

plt.show()

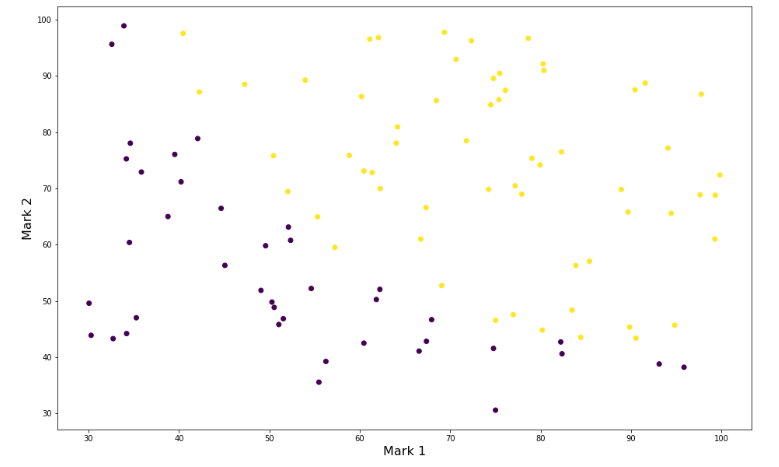


Рисунок 1 – график поступления студентов в университет

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

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def cost\_function(theta, X, y):

m = len(y)

h\_theta = sigmoid(np.dot(X, theta))

# J = (1 / m) \* ((-y' \* log(h\_theta)) - (1 - y)' \* log(1 - h\_theta));

J = (1 / m) \* ((np.dot(-y.T, np.log(h\_theta))) - np.dot((1 - y).T, np.log(1 - h\_theta)))

return J

def gradient(theta, X, y):

# grad = (1 / m) \* (h\_theta - y)' \* X;

m = len(y)

h\_theta = sigmoid(np.dot(X, theta))

return (1 / m) \* np.dot((h\_theta - y).T, X)

**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).**

**def** optim\_func(X, y):

**def** F(z):

W, b = z[:-1], z[-1]

model = LogisticRegression(W, b)

**return** model.loss(X, y)

**return** F

**def** optim\_func\_grad(X, y):

**def** G(z):

W, b = z[:-1], z[-1]

model = LogisticRegression(W, b)

dW, db = model.grad(X, y)

**return** np.array(dW.tolist() + [db])

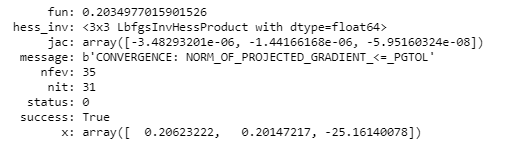
**return** G

z0 = get\_initialization(dim=X.shape[1], concat=**True**)

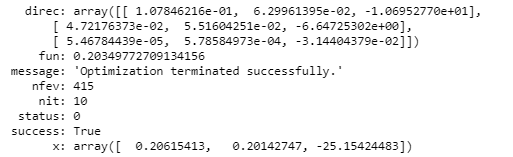
func = optim\_func(X, y)

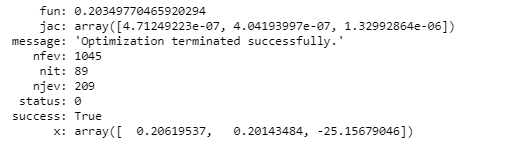
func\_grad = optim\_func\_grad(X, y)

minimize(func, z0, jac=func\_grad, method='L-BFGS-B')



minimize(func, z0, method='Powell')



minimize(func, z0, method='CG')

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

def h0x(X, theta):

return sigmoid(np.dot(X.T, theta))

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

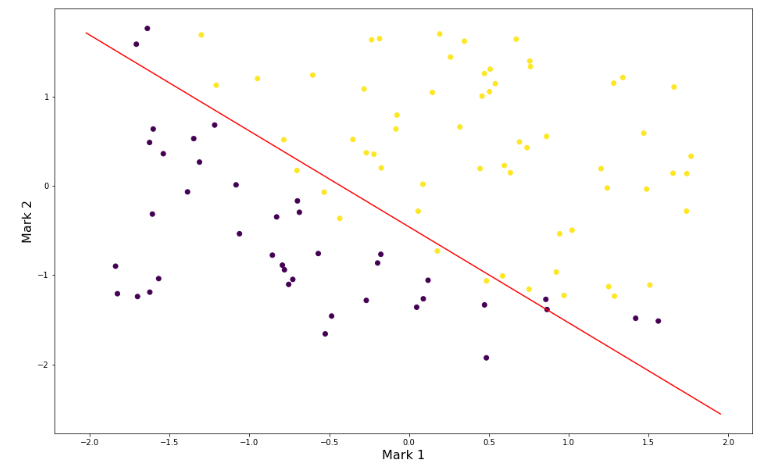


Рисунок 2 – график разделяющей прямой совмещенный с исходными данными

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

ex2data2 = np.loadtxt('Data/Lab 2/ex2data2.txt', delimiter=',') X, y = ex2data2[:, :2], ex2data2[:, 2]

**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.**

plt.figure(figsize=(16, 10))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

plt.xlabel('Test 1', size=16)

plt.ylabel('Test 2', size=16)

plt.show()

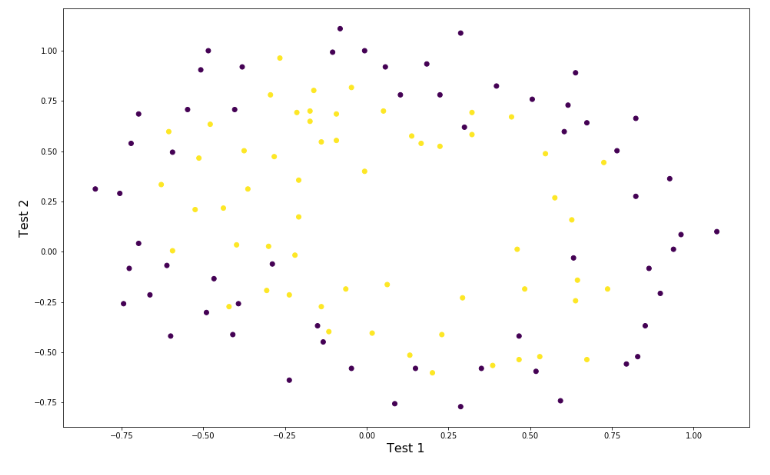


Рисунок 3 – график результатов тестов, где точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет

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

def polynom\_multi\_var(p1, p2):

def multiply(x): # 6 combination

return (x[0] \*\* p1) \* (x[1] \*\* p2)

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

# 9

map = {}

for i in range(0, 7):

for j in range(0, 7):

if i + j <= 6:

[key, fn] = polynom\_multi\_var(i, j)

map[key] = fn

# len(map.keys()) == 28

XX = []

for i in X.values:

a = []

for key in map.keys():

a.append(map[key](i))

XX.append(np.array(a))

X = np.array(XX)

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

**class** **L2RegualizedLogisticRegression**(LogisticRegression):

**def** \_\_init\_\_(self, W, b, C):

super().\_\_init\_\_(W, b)

self.C = C

**def** loss(self, X, y):

**return** super().loss(X, y) + 0.5 \* self.C \* np.sum(self.W\*\*2)

**def** grad(self, X, y):

dW, db = super().grad(X, y)

**return** dW + self.C \* self.W, db

W, b = get\_initialization(dim=X\_wide.shape[1])

regularized\_model = L2RegualizedLogisticRegression(W, b, 1e-4)

gradient\_descent = get\_gradient\_descent(regularized\_model, 0.1, 1e-6)

result = gradient\_descent(X\_combinations, y)

result['model']

**11. Реализуйте другие методы оптимизации.**

**def** optim\_func(X, y, C=1e-4):

**def** F(z):

W, b = z[:-1], z[-1]

model = L2RegualizedLogisticRegression(W, b, C)

**return** model.loss(X, y)

**return** F

**def** optim\_func\_grad(X, y, C=1e-4):

**def** G(z):

W, b = z[:-1], z[-1]

model = L2RegualizedLogisticRegression(W, b, C)

dW, db = model.grad(X, y)

**return** np.array(dW.tolist() + [db])

**return** G

z0 = get\_initialization(dim=X\_combinations.shape[1], concat=**True**)

func = optim\_func(X\_combinations, y)

func\_grad = optim\_func\_grad(X\_combinations, y)

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

**def** predict(X, model):

X\_combs = polynomial\_transform.transform(X)

**return** model(X\_combs)

predict(X[[0,30]], result['model'])

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

**def** plot\_data\_and\_model(X, y, model, figsize=(16,10)):

plt.figure(figsize=figsize)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

xx1 = np.linspace(-1, 1, 100)

xx2 = np.linspace(-1, 1, 100)

z = np.zeros((100, 100))

**for** i\_x1, x1 **in** enumerate(xx1):

**for** i\_x2, x2 **in** enumerate(xx2):

z[i\_x1, i\_x2] = predict(np.array([[x1, x2]]), model)

plt.contour(xx1, xx2, z, levels=[0.5], colors='r', linestyles='-')

plt.xlabel('Test 1', size=16)

plt.ylabel('Test 2', size=16)

plt.show()

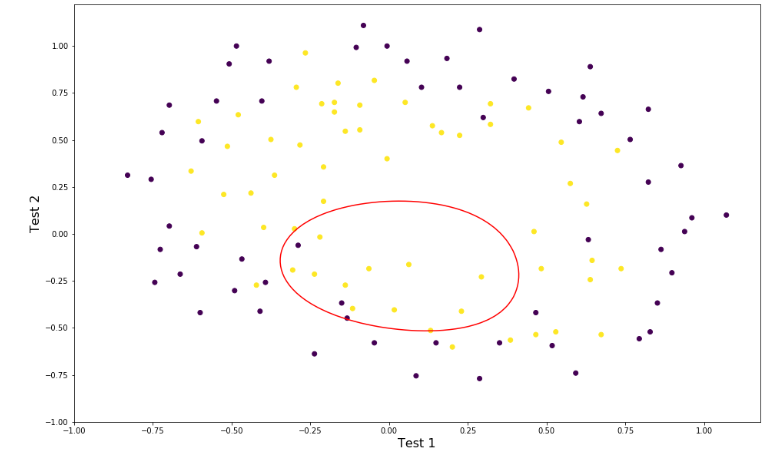


Рисунок 4 – график разделяющей совмещенный с исходными данными (lambda=0.1)

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

C\_list = [0, 1e-2, 1e-1, 0.25, 0.5, 1.]

**for** C **in** C\_list:

print('C = **{}**'.format(C))

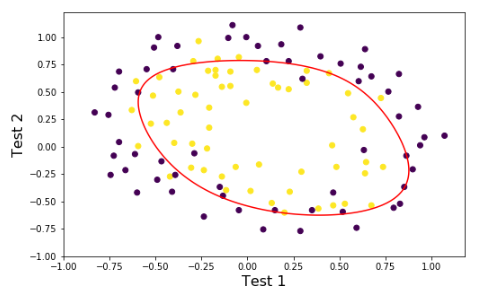
W, b = get\_initialization(dim=X\_wide.shape[1])

regularized\_model = L2RegualizedLogisticRegression(W, b, C)

gradient\_descent = get\_gradient\_descent(regularized\_model, 0.1, 1e-6)

result = gradient\_descent(X\_wide, y)

plot\_data\_and\_model(X, y, result['model'], figsize=(8,5))



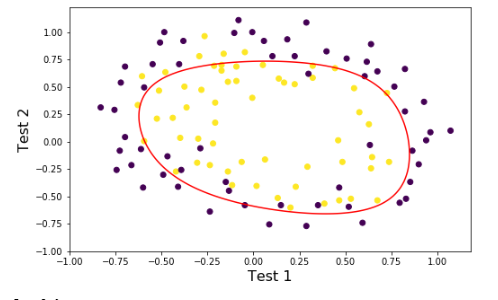
Рисунок 5 – Разделяющая кривая при lambda=0.001

Рисунок 6 – разделяющая кривая при lambda=0.5

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

ex2data3 = io.loadmat('Data/Lab 2/ex2data3.mat')

X, y = ex2data3['X'], ex2data3['y'].flatten()

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

n = 1

**for** idx **in** range(1, 11):

idx\_range = np.where(y == idx)[0]

idx\_subset = np.random.choice(idx\_range, size=n)

images = X[idx\_subset].reshape((n, 20, 20), order='F')

**for** batch\_idx **in** range(n):

plt.imshow(images[0, :, :], cmap='gray')

plt.show()



Рисунок 7 – визуализация чисел

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

X\_norm = X - X.mean(axis=0)

y\_new = y.copy()

y\_new[y == 5] = 1

y\_new[y != 5] = 0

model = LogisticRegression(\*get\_initialization(dim=X\_norm.shape[1]))

gradient\_descent = get\_gradient\_descent(model, 0.1, 1e-6)

result = gradient\_descent(X\_norm, y\_new)

**18. Добавьте L2-регуляризацию к модели.**

W, b = get\_initialization(dim=X\_norm.shape[1])

regularized\_model = L2RegualizedLogisticRegression(W, b, 5e-4)

gradient\_descent = get\_gradient\_descent(regularized\_model, 0.1, 1e-6)

result = gradient\_descent(X\_norm, y\_new)

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

**class** **OneVsRestClassifier**:

**def** \_\_init\_\_(self, base\_model, alpha, eps=1e-6):

self.\_base\_model = base\_model

self.\_alpha = alpha

self.\_eps = eps

self.\_trained\_models = OrderedDict()

**def** fit(self, X, y):

**for** class\_id **in** sorted(np.unique(y)):

print('Fitting **{}** vs all'.format(class\_id))

model = deepcopy(self.\_base\_model)

**assert** model != self.\_base\_model

y\_mocked = y.copy()

y\_mocked[y == class\_id] = 1

y\_mocked[y != class\_id] = 0

gradient\_descent = get\_gradient\_descent(model, self.\_alpha, self.\_eps)

result = gradient\_descent(X, y\_mocked)

self.\_trained\_models[class\_id] = result['model']

**def** predict(self, X):

**return** pd.DataFrame({class\_id: model(X) **for** class\_id, model **in** self.\_trained\_models.items()})

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

W, b = get\_initialization(dim=X\_norm.shape[1])

base\_model = L2RegualizedLogisticRegression(W, b, 1e-5)

one\_vs\_all\_model = OneVsRestClassifier(base\_model, alpha=1.0)

one\_vs\_all\_model.fit(X\_norm, y)

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

y\_pred\_probas = one\_vs\_all\_model.predict(X\_norm) y\_pred = y\_pred\_probas.idxmax(axis=1).to\_numpy() accuracy = np.sum(y == y\_pred) / len(y) print('Correct answers percent: **{:.2%}**'.format(accuracy))

Correct answers percent: 94.7%

**Вывод**

В отличие от обычной регрессии, в методе логистической регрессии не производится предсказание значения числовой переменной исходя из выборки исходных значений. Вместо этого, значением функции является вероятность того, что данное исходное значение принадлежит к определенному классу. Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено линейной границей (т.е. прямой) на две соответствующих классам области. В случае двух измерений — это просто прямая линия без изгибов. В случае трех — плоскость, и так далее. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма. Чтобы все работало, точки исходных данных должны разделяться линейной границей на две вышеупомянутых области. Если точки исходных данных удовлетворяют этому требованию, то их можно назвать линейно разделяемыми.