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

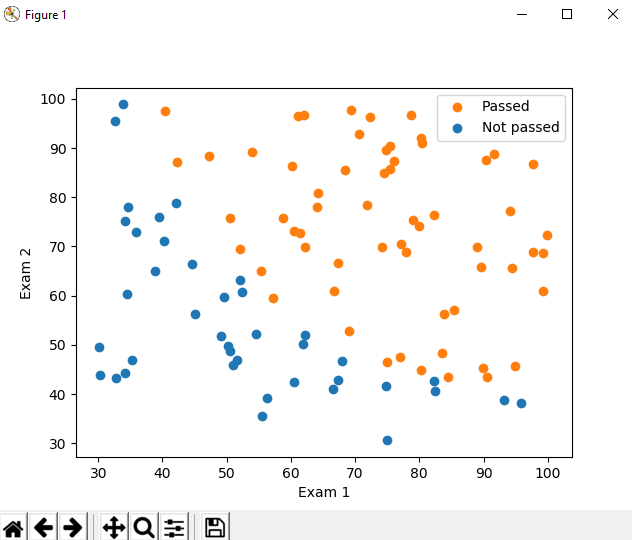
Минск 2019

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]

1. 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()



1. 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)

1. 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:

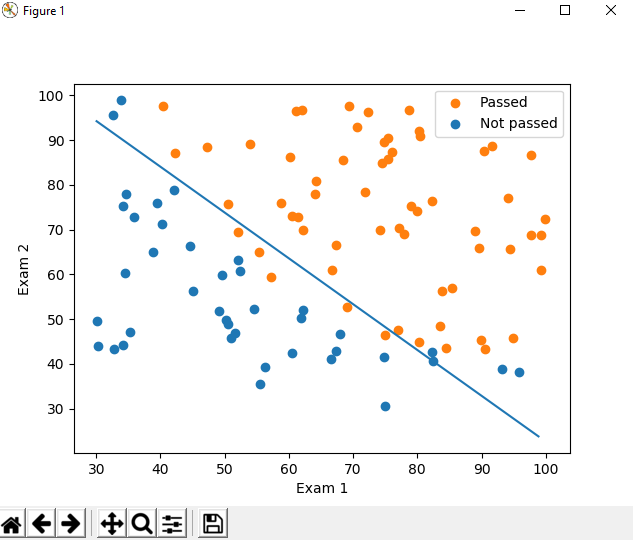
Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно:

1. 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

1. 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()

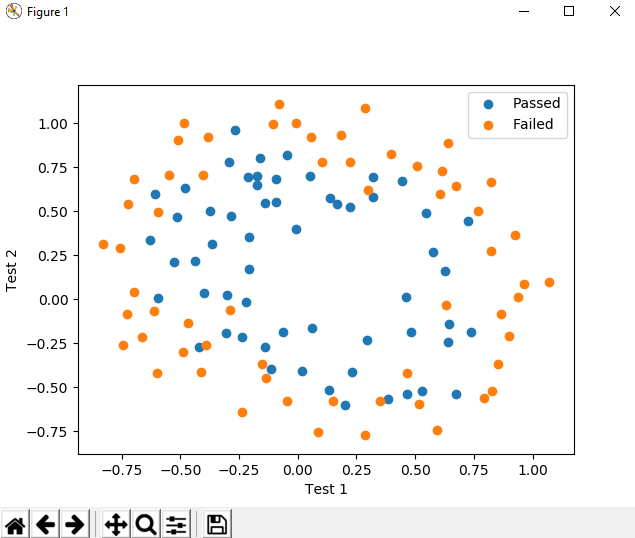


7) Загрузите данные **ex2data2.txt** из текстового файла.

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 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).

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)

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

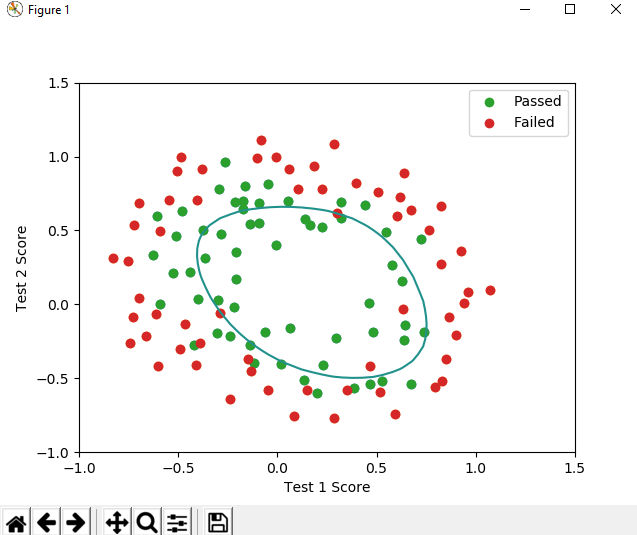
1. Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.

#12  
print('Predictions: ', sig(np.dot(X.values[0].T, theta\_bfgs)))

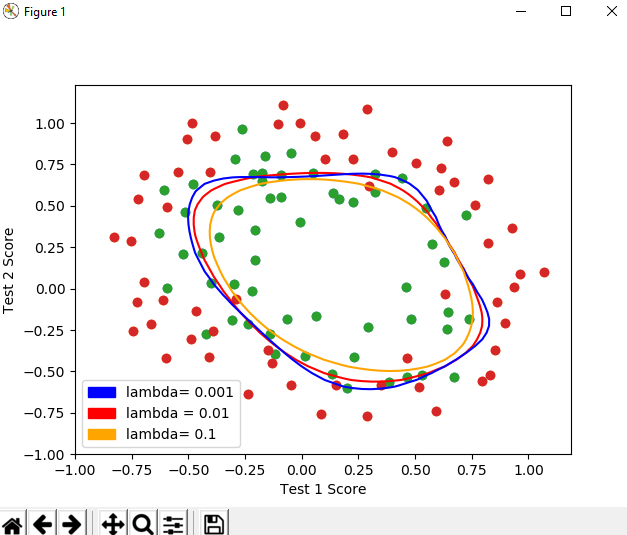
1. Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.

#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()



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

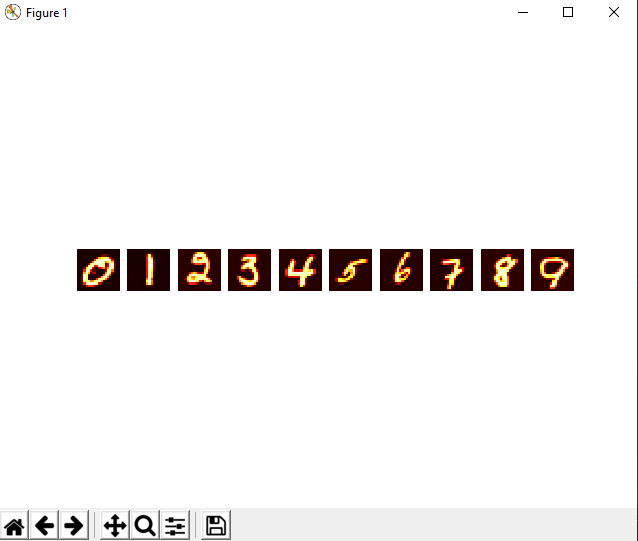


1. Загрузите данные **ex2data3.mat** из файла.

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

1. 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")



1. 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))

1. 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))

1. 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)

1. 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])

1. 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))

**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).