Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №2**

«Логистическая регрессия. Многоклассовая классификация»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2019

ХОД РАБОТЫ

**Задание.**

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.
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.
3. Реализуйте функции потерь J(θ) и градиентного спуска для логистической регрессии с использованием векторизации.
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).
5. Реализуйте функцию предсказания вероятности поступления студента в зависимости от значений оценок по экзаменам.
6. Постройте разделяющую прямую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 2.
7. Загрузите данные ex2data2.txt из текстового файла.
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.
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).
10. Реализуйте L2-регуляризацию для логистической регрессии и обучите ее на расширенном наборе признаков методом градиентного спуска.
11. Реализуйте другие методы оптимизации.
12. Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.
13. Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.
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.
16. Визуализируйте несколько случайных изображений из набора данных. Визуализация должна содержать каждую цифру как минимум один раз.
17. Реализуйте бинарный классификатор с помощью логистической регрессии с использованием векторизации (функции потерь и градиентного спуска).
18. Добавьте L2-регуляризацию к модели.
19. Реализуйте многоклассовую классификацию по методу “один против всех”.
20. Реализуйте функцию предсказания класса по изображению с использованием обученных классификаторов.
21. Процент правильных классификаций на обучающей выборке должен составлять около 95%.

**Результат выполнения:**

1. Код выгрузки данных из файла:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data1.txt')

data = pd.read\_csv(file\_path, header=None)

X = data.iloc[:, :-1] # first 2 column

y = data.iloc[:, 2] # last column

data.head()

2. График представлен ниже:

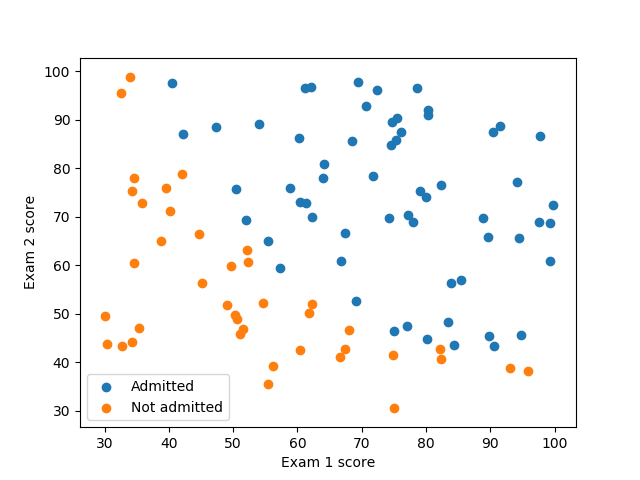
****

Рисунок 1 – график поступления студентов в университет

3. Код функции потерь и вычисления градиента:

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. Функция градиентного спуска:

# 4

temp = optimize.fmin\_tnc(

func=cost\_function,

x0=theta.flatten(),

fprime=gradient,

args=(X, y.flatten())

)

# the output of above function is a tuple whose first element contains the optimized values of theta

theta\_optimized = temp[0]

print(theta\_optimized)

temp = optimize.minimize(cost\_function, theta.flatten(), (X, y.flatten()), method='Nelder-Mead')

print(temp.x)

# Brovden Fletcher Goldfarb Shanno alghoritm

theta\_optimized = optimize.fmin\_bfgs(

cost\_function,

theta.flatten(),

gradient,

(X, y.flatten())

)

print(theta\_optimized)

Результат выполнения:

[-25.16131856 0.20623159 0.20147149]

[-25.16130062 0.20623142 0.20147143]

[-25.16133284 0.2062317 0.2014716 ]

5. Код функции:

def h0x(X, theta):

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

6. График представлен ниже:

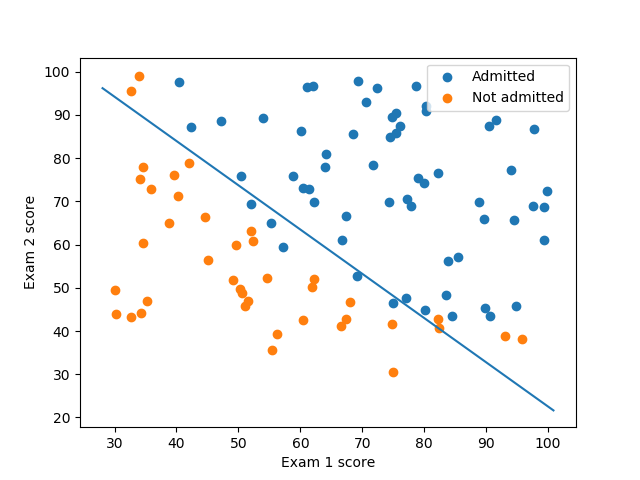


Рисунок 2 – график разделяющей прямой совмещенный с исходными данными

7. Код загрузки второго датасета:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data2.txt')

data = pd.read\_csv(file\_path, header=None)

X = data.iloc[:, :-1] # first 2 column

y = data.iloc[:, 2] # last column

data.head()

8. График приведён ниже:

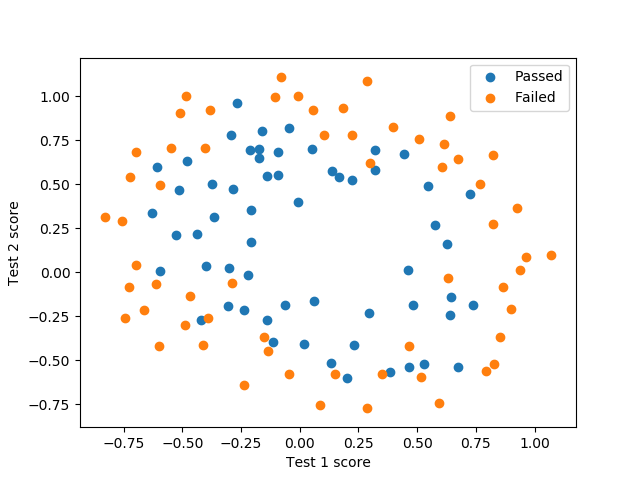


Рисунок 3 – график результатов тестов, где точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет

9. Код реализации:

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. Код реализации:

def cost\_function\_regularized(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, np.log(1 - h\_theta))) + (lambda\_ / (2 \* m)) \* np.sum(theta[1:]\*\*2)

return J

def gradient\_regularized(theta, X, y, lambda\_=0):

m = len(y)

grad = np.zeros([m, 1])

grad = (1 / m) \* np.dot(X.T, (sigmoid(np.dot(X, theta)) - y))

grad[1:] = grad[1:] + (lambda\_ / m) \* theta[1:]

return grad

# Set regularization parameter lambda to 1

lambda\_ = 0.1

(m, n) = X.shape

theta = np.zeros((n + 1, 1))

X = np.hstack((np.ones((m, 1)), X))

y = y[:, np.newaxis]

print('Cost at initial theta (zeros): %s', cost\_function\_regularized(theta, X, y, lambda\_)[0][0])

print('Expected cost (approx): 0.693')

output = optimize.fmin\_tnc(

func=cost\_function\_regularized,

x0=theta.flatten(),

fprime=gradient\_regularized,

args=(X, y.flatten(), lambda\_)

)

temp = output[0]

print('Reg fmin\_tnc: %s' % temp) # theta contains the optimized values

Здесь коэффициент лямбды установлен в 0.1, хотя на курсере он 1 – но при 0.1 Python просто не может посчитать.

11. Код реализации:

temp = optimize.minimize(cost\_function\_regularized, theta.flatten(), (X, y.flatten(), lambda\_), method='Nelder-Mead')

print('Nelder-Mead: %s' % temp.x)

theta\_optimized = optimize.fmin\_bfgs(

cost\_function\_regularized,

theta.flatten(),

gradient\_regularized,

(X, y.flatten(), lambda\_)

)

print('Brovden Fletcher Goldfarb Shanno alghoritm: %s' % theta\_optimized)

12. Код реализован в пункте номер 5.

13. График представлен ниже:

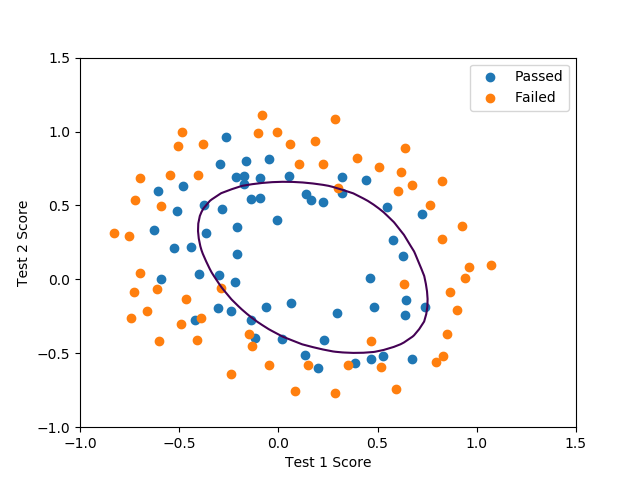


Рисунок 4 – график разделяющей совмещенный с исходными данными (lambda=0.1)

14. Графики представлены ниже

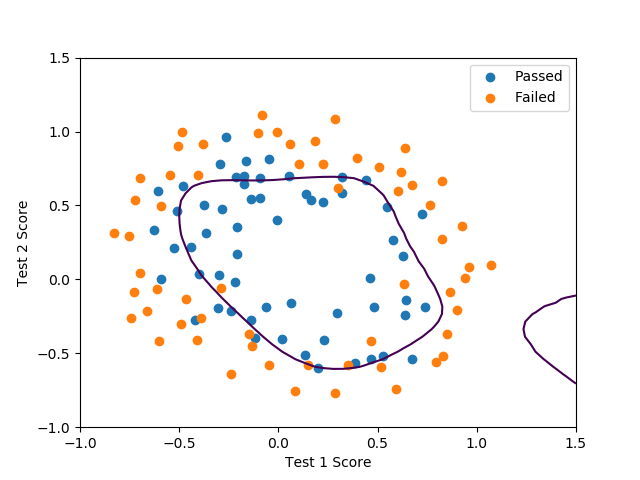


Рисунок 5 – Разделяющая кривая при lambda=0.001

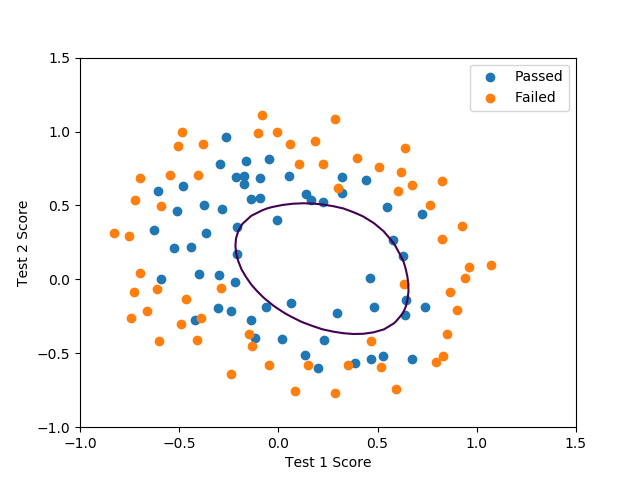


Рисунок 6 – разделяющая кривая при lambda=0.5

15. Код загрузки третьего датасета:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data3.mat')

data = sio.loadmat(file\_path)

X = data.get('X')

y = data.get('y')

16. Визуализация чисел с 0 до 9



Рисунок 7 – визуализация чисел

17. Код реализации:

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)) # initial parameters

print("Cost with zeros theta: ", cost\_function\_regularized(theta[0], X, y))

print("Gradient with zeros theta: ", gradient\_regularized(theta.T, X, y))

Результат выполнения (указан только один результат, потому что тета-вектор слишком большой, чтобы вставлять его в отчёт):

'Cost with zeros theta: ', array([-17.05142064])

18. Код реализации:

print("Cost with zeros theta: ", cost\_function\_regularized(theta[0], X, y, lambda\_))

print("Gradient with zeros theta: ", gradient\_regularized(theta.T, X, y, lambda\_))

Результат выполнения (указан только один результат, потому что тета-вектор слишком большой, чтобы вставлять его в отчёт):

'Cost with zeros theta: ', array([-17.05142064])

19. Код реализации:

for i in range(k):

digit\_class = i if i else 10

theta[i] = optimize.fmin\_cg(

f=cost\_function\_regularized,

x0=theta[i],

fprime=gradient\_regularized,

args=(X, (y == digit\_class).flatten().astype(np.int), lmbda),

maxiter=50

)

20. Код реализации:

def predict\_number(X, theta):

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

print("Predicted number: ", predict\_number(X[1490], theta), "Real: ", y[1490][0])

Результат выполнения:

'Predicted number: ', 2, 'Real: ', 2

21. Код реализации:

pred = np.argmax(np.dot(X, theta.T), axis=1)

pred = [e if e else 10 for e in pred] # convert 0 to 10

predictions = 0

for i in range(len(pred)):

if pred[i] == y[i][0]:

predictions += 1

print("Accuracy: ", (predictions / len(y)) \* 100)

Результат выполнения:

Accuracy: 95.12

**Программный код:**

from \_\_future\_\_ import division

from scipy import optimize

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import os

import scipy.io as sio

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 cost\_function\_regularized(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, np.log(1 - h\_theta))) + (lambda\_ / (2 \* m)) \* np.sum(theta[1:]\*\*2)

return J

def h0x(X, theta):

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

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]

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)

def gradient\_regularized(theta, X, y, lambda\_=0):

m = len(y)

grad = np.zeros([m, 1])

grad = (1 / m) \* np.dot(X.T, (sigmoid(np.dot(X, theta)) - y))

grad[1:] = grad[1:] + (lambda\_ / m) \* theta[1:]

return grad

def predict\_number(X, theta):

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

if \_\_name\_\_ == '\_\_main\_\_':

# 1

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data1.txt')

data = pd.read\_csv(file\_path, header=None)

X = data.iloc[:, :-1] # first 2 column

y = data.iloc[:, 2] # last column

data.head()

# 2

admitted = y == 1

failed = y != 1

adm = plt.scatter(X[admitted][0].values, X[admitted][1].values)

not\_adm = plt.scatter(X[failed][0].values, X[failed][1].values)

plt.xlabel('Exam 1 score')

plt.ylabel('Exam 2 score')

plt.legend((adm, not\_adm), ('Admitted', 'Not admitted'))

plt.show()

# 3

(m, n) = X.shape

X = np.hstack((np.ones((m, 1)), X))

y = y[:, np.newaxis]

theta = np.zeros((n + 1, 1)) # [[0.] [0.] [0.]]

print('Cost at initial theta (zeros): ', cost\_function(theta, X, y)[0][0])

# test (from an Octave)

print('Expected gradients (approx): [-0.1000, -12.0092, -11.2628]')

print('Real gradient: %s' % gradient(theta, X, y))

# Compute and display cost and gradient with non-zero theta

test\_theta = np.array([[-24], [.2], [.2]])

print('Expected cost (approx): 0.218')

print('Cost at test theta: %s' % cost\_function(test\_theta, X, y)[0][0])

# 4

temp = optimize.fmin\_tnc(

func=cost\_function,

x0=theta.flatten(),

fprime=gradient,

args=(X, y.flatten())

)

# the output of above function is a tuple whose first element contains the optimized values of theta

theta\_optimized = temp[0]

print(theta\_optimized)

temp = optimize.minimize(cost\_function, theta.flatten(), (X, y.flatten()), method='Nelder-Mead')

print(temp.x)

# Brovden Fletcher Goldfarb Shanno alghoritm

theta\_optimized = optimize.fmin\_bfgs(

cost\_function,

theta.flatten(),

gradient,

(X, y.flatten())

)

print(theta\_optimized)

# 5

print('h0x test')

print(h0x(np.array([1, 34.62365962451697, 78.0246928153624]), theta\_optimized))

# 6

plot\_x = [np.min(X[:, 1] - 2), np.max(X[:, 2] + 2)]

plot\_y = -1 / theta\_optimized[2]\*(theta\_optimized[0] + np.dot(theta\_optimized[1], plot\_x))

mask = y.flatten() == 1

adm = plt.scatter(X[mask][:, 1], X[mask][:, 2])

not\_adm = plt.scatter(X[~mask][:, 1], X[~mask][:, 2])

decision\_boundary = plt.plot(plot\_x, plot\_y)

plt.xlabel('Exam 1 score')

plt.ylabel('Exam 2 score')

plt.legend((adm, not\_adm), ('Admitted', 'Not admitted'))

plt.show()

# 7

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data2.txt')

data = pd.read\_csv(file\_path, header=None)

X = data.iloc[:, :-1] # first 2 column

y = data.iloc[:, 2] # last column

data.head()

# 8

passed = y == 1

failed = y != 1

psd = plt.scatter(X[passed][0].values, X[passed][1].values)

not\_psd = plt.scatter(X[failed][0].values, X[failed][1].values)

plt.xlabel('Test 1 score')

plt.ylabel('Test 2 score')

plt.legend((psd, not\_psd), ('Passed', 'Failed'))

plt.show()

# 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

# Set regularization parameter lambda to 1

lambda\_ = 0.1

(m, n) = X.shape

theta = np.zeros((n + 1, 1))

X = np.hstack((np.ones((m, 1)), X))

y = y[:, np.newaxis]

print('Cost at initial theta (zeros): %s', cost\_function\_regularized(theta, X, y, lambda\_)[0][0])

print('Expected cost (approx): 0.693')

output = optimize.fmin\_tnc(

func=cost\_function\_regularized,

x0=theta.flatten(),

fprime=gradient\_regularized,

args=(X, y.flatten(), lambda\_)

)

temp = output[0]

print('Reg fmin\_tnc: %s' % temp) # theta contains the optimized values

# 11

temp = optimize.minimize(cost\_function\_regularized, theta.flatten(), (X, y.flatten(), lambda\_), method='Nelder-Mead')

print('Nelder-Mead: %s' % temp.x)

theta\_optimized = optimize.fmin\_bfgs(

cost\_function\_regularized,

theta.flatten(),

gradient\_regularized,

(X, y.flatten(), lambda\_)

)

print('Brovden Fletcher Goldfarb Shanno alghoritm: %s' % theta\_optimized)

# 12

print(h0x(X[0], theta\_optimized))

print(h0x(X[0], temp.x))

print(h0x(X[0], output[0]))

# 13

u = np.linspace(-1, 1.5, 50)

v = np.linspace(-1, 1.5, 50)

z = np.zeros((len(u), len(v)))

for i in range(len(u)):

for j in range(len(v)):

a = [1]

for key in map.keys():

a.append(map[key]([u[i], v[j]]))

z[i, j] = h0x(np.array(a), theta\_optimized)

mask = y.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(u, v, z, 0)

plt.xlabel('Test 1 Score')

plt.ylabel('Test 2 Score')

plt.legend((passed, failed), ('Passed', 'Failed'))

plt.show()

# 14

# TODO: implement charts for different lambda

X = np.array(XX)

X = np.hstack((np.ones((m, 1)), X))

(m, n) = X.shape

correct\_identified = 0

for i in range(m):

if round(h0x(X[i], theta\_optimized)) == y[i]:

correct\_identified += 1

print("Correct recognition(%): ", correct\_identified / m)

# 15

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex2data3.mat')

data = sio.loadmat(file\_path)

X = data.get('X')

y = data.get('y')

# 16

images = {}

for i in range(len(y)):

images[y[i][0]] = i # assign the latest index of number image

keys = images.keys()

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

for j in range(len(keys)):

# reshape back to 20 pixel by 20 pixel

axis[j].imshow(X[images.get(images.keys()[j]), :].reshape(20, 20, order="F"), cmap="hot")

axis[j].axis("off")

plt.show()

# 17

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)) # initial parameters

print("Cost with zeros theta: ", cost\_function\_regularized(theta[0], X, y))

# print("Gradient with zeros theta: ", gradient\_regularized(theta.T, X, y, lambda\_))

# 18

print("Cost with zeros theta: ", cost\_function\_regularized(theta[0], X, y, lambda\_))

# print("Gradient with zeros theta: ", gradient\_regularized(theta.T, X, y))

# 19

for i in range(k):

digit\_class = i if i else 10

theta[i] = optimize.fmin\_cg(

f=cost\_function\_regularized,

x0=theta[i],

fprime=gradient\_regularized,

args=(X, (y == digit\_class).flatten().astype(np.int), lmbda),

maxiter=50

)

# 20

print("Predicted number: ", predict\_number(X[1490], theta), "Real: ", y[1490][0])

# 21

pred = np.argmax(np.dot(X, theta.T), axis=1)

pred = [e if e else 10 for e in pred] # convert 0 to 10

predictions = 0

for i in range(len(pred)):

if pred[i] == y[i][0]:

predictions += 1

print("Accuracy: ", (predictions / len(y)) \* 100)