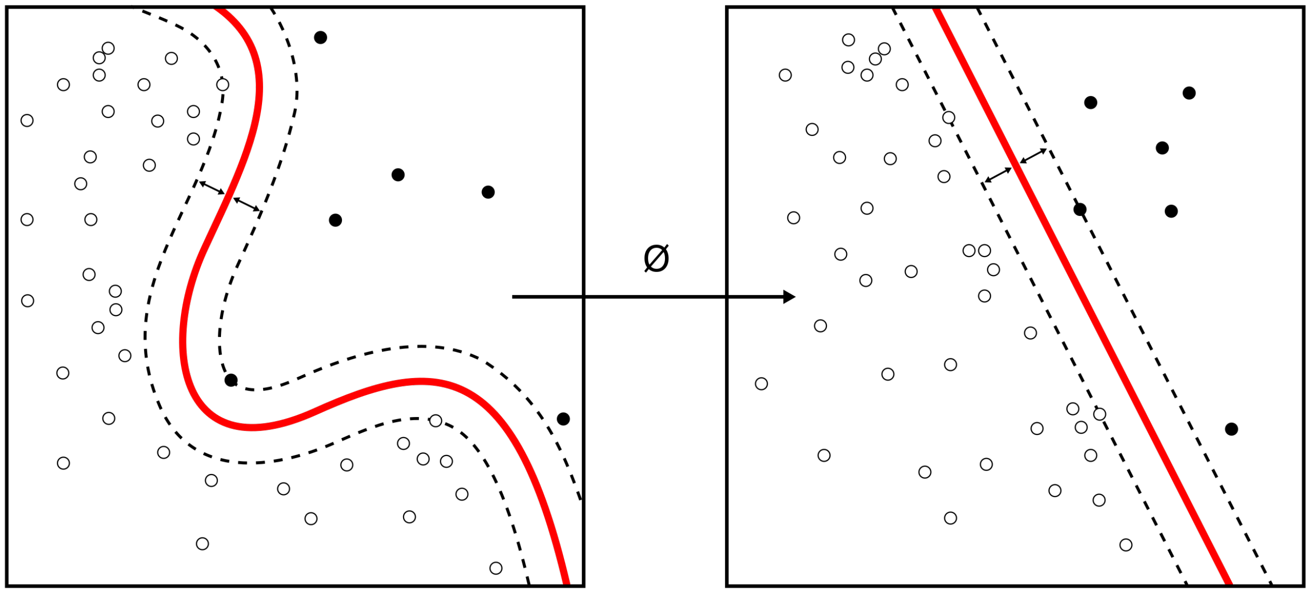


Classification

CS 520 Final Question 3

Spring 2019

Xuenan Wang

NET ID: xw336

CLASSIFICATION

a) Construct a model to classify images as Class A or B, and train it on the indicated data. Specify your trained model. What does your model predict for each of the unlabeled images? Give the details of your model, its training, and the final result. Do the predictions make sense, to you?

Model: Binary Logistic Regression

Sigmoid Function: $h(z) = \dfrac{1}{1+e^{-z}}$

Cost Function: $Cost(h_w^x, y) = \begin{cases} \log\ h_w(x) & \text{if } y = 1 \\ \log(1-h(x)) & \text{if } y = 0 \end{cases}$

Gradient Ascent: $w_j = w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)}))x_j^{(i)}$

Feature:

1: $Average(\sqrt{(x^2-0^2)+(y^2-0^2)})$

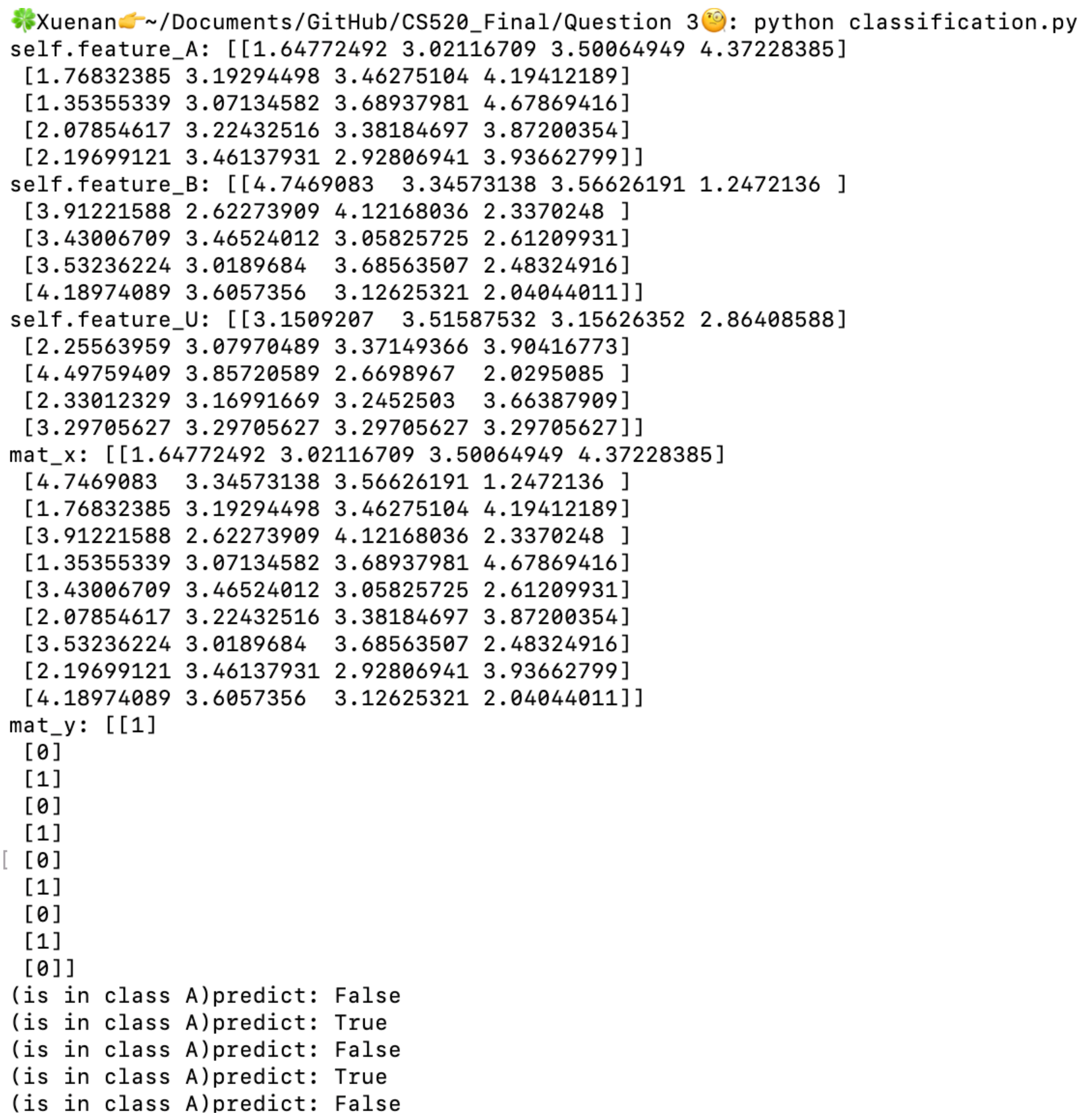
2: $Average(\sqrt{(x^2-4^2)+(y^2-0^2)})$

3: $Average(\sqrt{(x^2-0^2)+(y^2-4^2)})$

4: $Average(\sqrt{(x^2-4^2)+(y^2-4^2)})$

Result: Shown as Figure 1.

Analysis: I think this prediction is reasonable. The features I have are Euclidian Distance that each point to four corner points. As shown in Figure 2. This feature can show the relative positions of training data. To be precise, is original data tended to one corner or not. As we can clearly see that the training data of class A is near to upper-left corner while class B is near to down-right corner. Logistic Regression is known as its simplicity and good performance in binary classification scenario. Considering the training data is not complex and it's not much, choosing a complex model will lead to unnecessary waste of time and energy, also with more complex model, the overfitting risk is more serious than a simple one.



```
🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 3🧐: python classification.py
self.feature_A: [[1.64772492 3.02116709 3.50064949 4.37228385]
 [1.76832385 3.19294498 3.46275104 4.19412189]
 [1.35355339 3.07134582 3.68937981 4.67869416]
 [2.07854617 3.22432516 3.38184697 3.87200354]
 [2.19699121 3.46137931 2.92806941 3.93662799]]
self.feature_B: [[4.7469083  3.34573138 3.56626191 1.2472136 ]
 [3.91221588 2.62273909 4.12168036 2.3370248 ]
 [3.43006709 3.46524012 3.05825725 2.61209931]
 [3.53236224 3.0189684  3.68563507 2.48324916]
 [4.18974089 3.6057356  3.12625321 2.04044011]]
self.feature_U: [[3.1509207  3.51587532 3.15626352 2.86408588]
 [2.25563959 3.07970489 3.37149366 3.90416773]
 [4.49759409 3.85720589 2.6698967  2.0295085 ]
 [2.33012329 3.16991669 3.2452503  3.66387909]
 [3.29705627 3.29705627 3.29705627 3.29705627]]
mat_x: [[1.64772492 3.02116709 3.50064949 4.37228385]
 [4.7469083  3.34573138 3.56626191 1.2472136 ]
 [1.76832385 3.19294498 3.46275104 4.19412189]
 [3.91221588 2.62273909 4.12168036 2.3370248 ]
 [1.35355339 3.07134582 3.68937981 4.67869416]
 [3.43006709 3.46524012 3.05825725 2.61209931]
 [2.07854617 3.22432516 3.38184697 3.87200354]
 [3.53236224 3.0189684  3.68563507 2.48324916]
 [2.19699121 3.46137931 2.92806941 3.93662799]
 [4.18974089 3.6057356  3.12625321 2.04044011]]
mat_y: [[1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]]
(is in class A)predict: False
(is in class A)predict: True
(is in class A)predict: False
(is in class A)predict: True
(is in class A)predict: False
```

Figure 1. Running result.

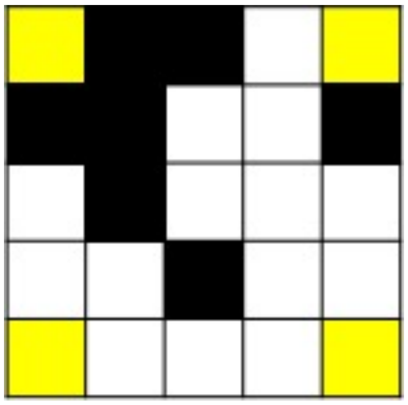


Figure 2. Example of features.

b) The data provided is quite small, and overfitting is a serious risk. What steps can you take to avoid it?

Since the data provided is quite small, I chose a relative single model to do the classification. The two best way to avoid overfitting is to enlarge the data and use a simple model. We can not enlarge the data set in this case, therefore, choosing a simple model should be our best choice. Also, we can try to train the data in less times, regularize the data and we probably can add some noice to the data.

c) Construct and train a second type of model. Specify its details. How do its predictions compare to the first model? Are there any differences, and what about the two models caused the differences?

Model: k-nearest neighbors algorithm.

Features: same as above.

I noticed that the data after euclidian distance calculation is regular, thus I did not normalize them considering they are calculated under same situation and method. Here the a working K-value is 5.5.

Result: Shown as Figure 3.

```
array_a: [5.335661033804954, 5.432000876847256, 5.43649606644924, 5.479554674922998, 5.414622381914055]
array_b: [6.905975271959061, 6.442784118190045, 5.978150725729, 6.136685442166315, 6.509159484807367]
array_u: [5.925855699560144, 5.46293230632822, 6.653138113917737, 5.447327322211816, 5.992394889640182]
numbers in class A: 5
result_u: ['B', 'A', 'B', 'A', 'B']
```

Figure 3. Running result of knn.

We can see from result that the classification is same as logistic regression model. This result make sense since we are using the same features, so the basic idea of this classification is the same. The only difference here is the way to classify these features. In logistic regression, we use gradient ascent to train the model, here our idea is by calculating euclidian distance to see how these features near to each other in four-dimensional space(we have four features, therefore we will need to calculate euclidian distance in four-dimensional space). And fo course, the result, as mentioned before, is not surprising.

APPENDIX: SOURCE CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# @Date    : 2019-05-13 20:14:00
# @Author  : Xuenan(Roderick) Wang
# @Email   : roderick_wang@outlook.com
# @Github  : https://github.com/hello-roderickwang

import numpy as np
import matplotlib.pyplot as plt
import math

class Classifier:
	def __init__(self):
		self.class_A = np.array([])
		self.class_B = np.array([])
		self.class_U = np.array([])
		self.feature_A = np.zeros((5, 4))
		self.feature_B = np.zeros((5, 4))
		self.feature_U = np.zeros((5, 4))
		self.mat_x = []
		self.mat_y = []
		self.mat_u = []

	def sigmoid(self, x):
		return 1.0/(1.0+np.exp(-x))

	def logic_regression(self, alpha = 0.01, iter = 500):
		x = self.mat_x
		y = self.mat_y
		sample_num, feature_num = np.shape(x)
		weights = np.ones((feature_num, 1))
		for i in range(iter):
			fx = x*weights
			hx = self.sigmoid(fx)
			weights += alpha*x.T*(y-hx)
		return weights

	def get_accuracy(self, weights):
		u = self.mat_u
		# y = self.mat_y
		sample_num, feature_num = np.shape(u)
		accuracy = 0.0
		for i in range(sample_num):
			predict = 
self.sigmoid(u[i, :]*weights)[0, 0] > 0.5
			print('(is in class A)predict:', 
predict)
			# if predict == bool(y[i, 0]):
			#		accuracy += 1
		# print('accuracy:{0}%'.format(accuracy/
sample_num*100))

	def load_data(self):
		file_A = np.loadtxt('./Data/ClassA.txt')
		file_B = np.loadtxt('./Data/ClassB.txt')
		file_U = np.loadtxt('./Data/Mystery.txt')
		self.class_A = np.array(np.split(file_A, 5))
		self.class_B = np.array(np.split(file_B, 5))
		self.class_U = np.array(np.split(file_U, 5))

	def normalize(self, x):
		min_x = min(x)
		max_x = max(x)
		for i in range(len(x)):
			x[i] = (x[i]-min_x)/(max_x-min_x)
		return x

	def knn(self, k):
		array_a = []
		array_b = []
		array_u = []
		for i in range(5):

array_a.append(math.sqrt(self.feature_A[i]
[0]**2+self.feature_A[i][1]**2+self.feature_A[i]
[2]**2+self.feature_A[i][3]))

array_b.append(math.sqrt(self.feature_B[i]
[0]**2+self.feature_B[i][1]**2+self.feature_B[i]
[2]**2+self.feature_B[i][3]))

array_u.append(math.sqrt(self.feature_U[i]
[0]**2+self.feature_U[i][1]**2+self.feature_U[i]
[2]**2+self.feature_U[i][3]))
		print('array_a:', array_a)
		print('array_b:', array_b)
		print('array_u:', array_u)
		array = []
		for i in range(10):
			if i < 5:

array.append(array_a[i])
			else:

array.append(array_b[i-5])
		# print('array:',array)
		# array_n = self.normalize(array)
		# print('array:', array)
		ctr = 0
		for i in range(10):
			if array[i]<k:
				ctr += 1
		print('numbers in class A:', ctr)
		ctr_u = 0
		result_u = []
		for i in range(5):
			if array_u[i] < k:
				ctr_u += 1
				result_u.append('A')
			else:
				result_u.append('B')
		print('result_u:', result_u)

	def get_feature(self):
		for i in range(5):
			array_lu = []
			array_ru = []
			array_ld = []
			array_rd = []
			for x in range(5):
```

```
                for y in range(5):
                    if
self.class_A[i][x][y] == 1:

array_lu.append(math.sqrt((x-0)**2+(y-0)**2))

array_ru.append(math.sqrt((x-4)**2+(y-0)**2))

array_ld.append(math.sqrt((x-0)**2+(y-4)**2))

array_rd.append(math.sqrt((x-4)**2+(y-4)**2))
            sum = np.zeros(4)
            for j in range(len(array_lu)):
                sum[0] += array_lu[j]
                sum[1] += array_ru[j]
                sum[2] += array_ld[j]
                sum[3] += array_rd[j]
            self.feature_A[i][0] = sum[0]/
len(array_lu)
            self.feature_A[i][1] = sum[1]/
len(array_lu)
            self.feature_A[i][2] = sum[2]/
len(array_lu)
            self.feature_A[i][3] = sum[3]/
len(array_lu)
        print('self.feature_A:', self.feature_A)
        for i in range(5):
            array_lu = []
            array_ru = []
            array_ld = []
            array_rd = []
            for x in range(5):
                for y in range(5):
                    if
self.class_B[i][x][y] == 1:

array_lu.append(math.sqrt((x-0)**2+(y-0)**2))

array_ru.append(math.sqrt((x-4)**2+(y-0)**2))

array_ld.append(math.sqrt((x-0)**2+(y-4)**2))

array_rd.append(math.sqrt((x-4)**2+(y-4)**2))
            sum = np.zeros(4)
            for j in range(len(array_lu)):
                sum[0] += array_lu[j]
                sum[1] += array_ru[j]
                sum[2] += array_ld[j]
                sum[3] += array_rd[j]
            self.feature_B[i][0] = sum[0]/
len(array_lu)
            self.feature_B[i][1] = sum[1]/
len(array_lu)
            self.feature_B[i][2] = sum[2]/
len(array_lu)
            self.feature_B[i][3] = sum[3]/
len(array_lu)
        print('self.feature_B:', self.feature_B)
        for i in range(5):
            array_lu = []
            array_ru = []
            array_ld = []
            array_rd = []
            for x in range(5):
                for y in range(5):
                    if
self.class_U[i][x][y] == 1:

array_lu.append(math.sqrt((x-0)**2+(y-0)**2))

array_ru.append(math.sqrt((x-4)**2+(y-0)**2))

array_ld.append(math.sqrt((x-0)**2+(y-4)**2))

array_rd.append(math.sqrt((x-4)**2+(y-4)**2))
            sum = np.zeros(4)
            for j in range(len(array_lu)):
                sum[0] += array_lu[j]
                sum[1] += array_ru[j]
                sum[2] += array_ld[j]
                sum[3] += array_rd[j]
            self.feature_U[i][0] = sum[0]/
len(array_lu)
            self.feature_U[i][1] = sum[1]/
len(array_lu)
            self.feature_U[i][2] = sum[2]/
len(array_lu)
            self.feature_U[i][3] = sum[3]/
len(array_lu)
        print('self.feature_U:', self.feature_U)
        for i in range(5):

self.mat_x.append(self.feature_A[i][:])
            self.mat_y.append(1)

self.mat_x.append(self.feature_B[i][:])
            self.mat_y.append(0)

self.mat_u.append(self.feature_U[i][:])
        self.mat_x = np.mat(self.mat_x)
        self.mat_y = np.mat(self.mat_y).T
        self.mat_u = np.mat(self.mat_u)
        print('mat_x:', self.mat_x)
        print('mat_y:', self.mat_y)

if __name__ == '__main__':
    modle = Classifier()
    # modle.get_feature()
    modle.load_data()
    modle.get_feature()
    weights = modle.logic_regression(alpha = 0.01, iter
= 500)
    modle.get_accuracy(weights)
    modle.knn(5.5)
```