



TỐI ƯU HÓA NÂNG CAO PROJECT GIỮA KỲ

Giảng viên: Ts. Trần Nam Dũng

Học viên(Nhóm 3) - Đặng Ngọc Uyên
- Nguyễn Duy Linh
- Nguyễn Đức Trường

Cấu trúc bài thuyết trình



Phần I

Giới thiệu bài toán, dữ liệu và xử lý dữ liệu

Phần II

Các thuật toán của bài toán

Phần III

Nhận xét và kết luận

Phần I: Giới thiệu bài toán

Bài toán dự đoán xác suất 1 người có khả năng chết hay không trong thời gian đại dịch Covid_19

Dữ liệu được lấy ngẫu nhiên tại 1 số Quốc gia lân cận Trung Quốc

1. Giới thiệu dữ liệu

	id	case_in_country	reporting_date	Unnamed: 3	summary	location	country	gender	age	visiting Wuhan	from Wuhan	death	symptom
0	1	NaN	1/20/2020	NaN	new confirmed COVID-19 patient in Vietnam: 3 m...	Vinh Phuc	Vietnam	NaN	0.25	0	0	0	NaN
1	2	NaN	1/20/2020	NaN	new confirmed COVID-19 patient in Singapore: m...	Singapore	Singapore	male	0.50	0	0	0	NaN
2	3	NaN	1/21/2020	NaN	new confirmed COVID-19 patient in Singapore: 1...	Singapore	Singapore	male	1.00	0	0	0	NaN
3	4	NaN	1/21/2020	NaN	new confirmed imported COVID-19 pneumonia pati...	Hechi, Guangxi	China	female	2.00	1	0	0	NaN
4	5	NaN	1/21/2020	NaN	new confirmed COVID-19 patient in Malaysia: ma...	Johor	Malaysia	male	2.00	0	0	0	NaN
...
838	842	47.0	2/27/2020	NaN	new confirmed COVID-19 patient in South Korea:...	South Korea	South Korea	male	74.00	0	0	1	NaN
839	843	48.0	2/27/2020	NaN	new confirmed COVID-19 patient in Japan: male,...	Tokyo	Japan	male	85.00	0	0	1	fever, pneumonia
840	844	49.0	2/28/2020	NaN	new confirmed COVID-19 patient in South Korea:...	South Korea	South Korea	male	75.00	0	0	1	NaN
841	845	50.0	2/28/2020	NaN	new confirmed COVID-19 patient in Japan: male,...	Hokkaido	Japan	male	85.00	0	0	1	difficulty breathing
842	846	51.0	2/28/2020	NaN	new confirmed COVID-19 patient in Japan: male,...	Japan	Japan	male	75.00	0	0	1	cold, fever, pneumonia

843 rows x 13 columns

1. Giới thiệu dữ liệu và xử lý dữ liệu

- Chuyển location về kinh độ và vĩ độ
- Xử lý category data(gender và symptom)
- Chuẩn hóa dữ liệu về dạng chuẩn

```
feature = list(set(list(data.columns))-set(["death"]))
x = data[feature].values
y = data['death']
x=sn.fit_transform(x)
x0 = np.full((x.shape[0],1),1)
X = np.vstack([np.hstack([x0, x])])
D = X.shape[1]
N = X.shape[0]
```

```
# geolocator = Nominatim(user_agent="covid19")
# print(len(set(locationList)))
# lat = {}
# long = {}
# for i in set(locationList):
#     # print(i)
#     location = geolocator.geocode(i)
#     lat[i] = location.latitude
#     long[i] = location.longitude
```

```
if(data['gender'][j] == "male"):
    data['gender'][j] = 1
elif(data['gender'][j] == "female"):
    data['gender'][j] = 0
else:
    data['gender'][j] = -1
if(str(symptomList[j]) == "nan"):
    data['symptom'][j] = 0
else:
    lst = str(symptomList[j]).split(",")
    data['symptom'][j] = len(lst)
```

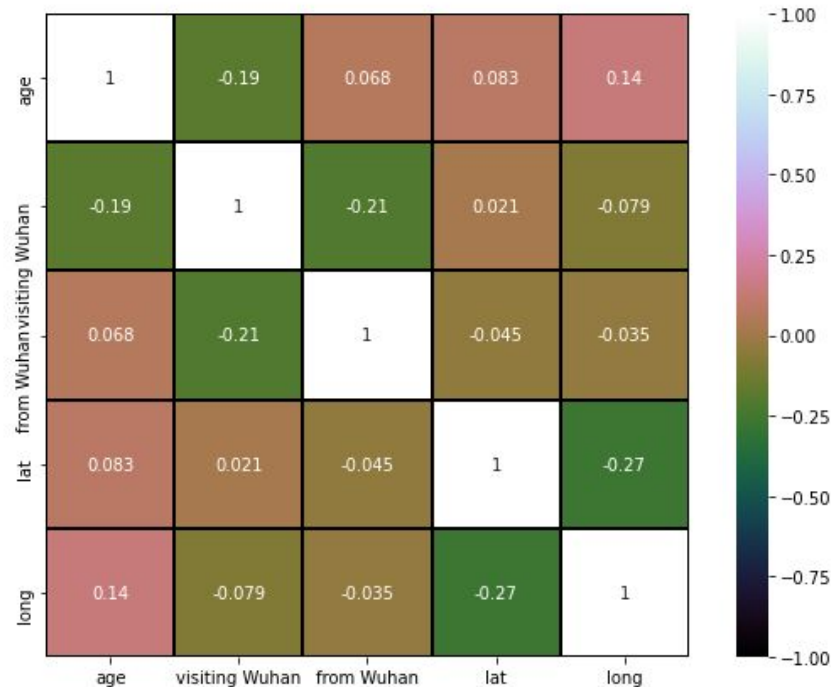
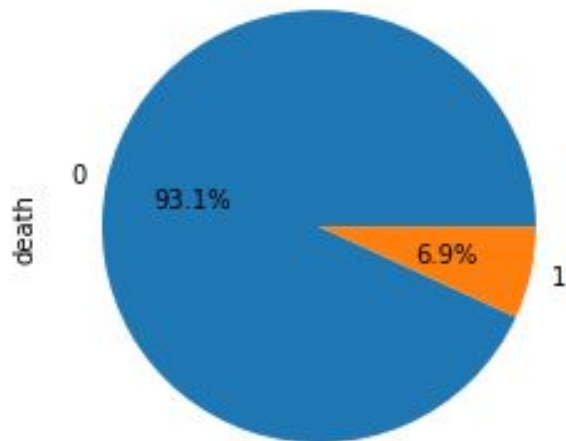
1. Giới thiệu dữ liệu và xử lý dữ liệu

	gender	age	visiting Wuhan	from Wuhan	death	symptom	lat	long
0	-1	0.25	0	0	0	0	13.290403	108.426511
1	1	0.50	0	0	0	0	1.357107	103.819499
2	1	1.00	0	0	0	0	1.357107	103.819499
3	0	2.00	1	0	0	0	35.000074	104.999927
4	1	2.00	0	0	0	0	4.569375	102.265682
...
838	1	74.00	0	0	1	0	36.638392	127.696119
839	1	85.00	0	0	1	2	36.574844	139.239418
840	1	75.00	0	0	1	0	36.638392	127.696119
841	1	85.00	0	0	1	1	36.574844	139.239418
842	1	75.00	0	0	1	3	36.574844	139.239418

343 rows x 8 columns

1. Giới thiệu dữ liệu

Nhận xét: Các biến đầu vào giường như không có tương quan với nhau



2. Các thuật toán



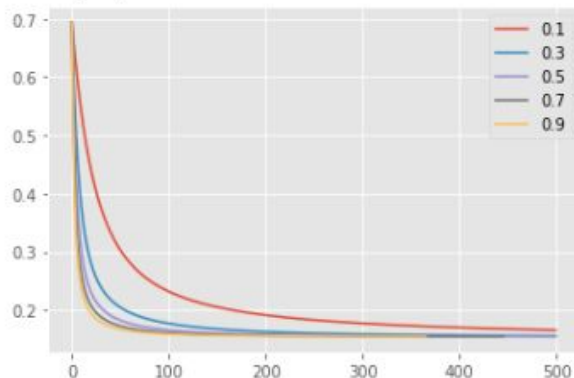
Nhóm sẽ trình bày về các thuật toán sau:

- BGD, SGD, Minibatch GD, BGD với backtracking
- Newton, NAG(Nesterov's Accelerated Gradient Descent), FISTA, NAG với backtracking
- Cyclic Coordinate Gradient Descent , randomized Coordinate Gradient Descent

2. CÁC THUẬT TOÁN

Dùng BGD và BGD backtracking

voi step_size Thời gian chạy là 0.1 0.3428634960000636
so vong lap can la 499
voi step_size Thời gian chạy là 0.3 0.3494614339997497
so vong lap can la 499
voi step_size Thời gian chạy là 0.5 0.3252556120000918
so vong lap can la 499
voi step_size Thời gian chạy là 0.7 0.2947225890002301
so vong lap can la 445
voi step_size Thời gian chạy là 0.9 0.24679093899976579
so vong lap can la 365



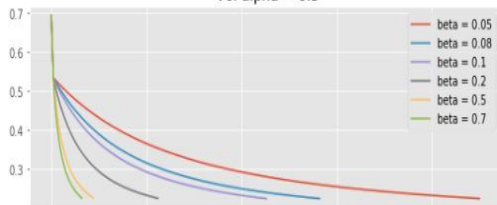
BGD với hệ số lr khác nhau

```
] steplist=[0.1, 0.3, 0.5, 0.7, 0.9]
for step in steplist:
    start = timeit.default_timer()
    pars1, cost1, iterts1 = gradient_descent_step_fixed(X, y, params_init, step, 500)
    stop = timeit.default_timer()
    print('voi step_size', "Thời gian chạy là", step, stop - start, "\n" "so vong lap can la", iterts1)
    label = ("step size" + str(step))
    plt.plot(np.arange(len(cost1)), cost1, label=label)
    plt.legend(["0.1", "0.3", "0.5", "0.7", "0.9"])
```

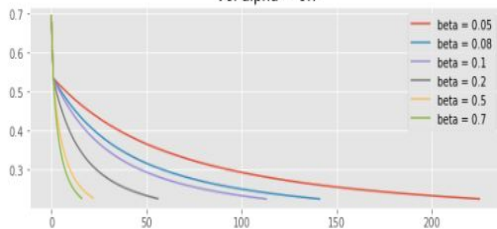
2. CÁC THUẬT TOÁN

Dùng BGD và BGD backtracking

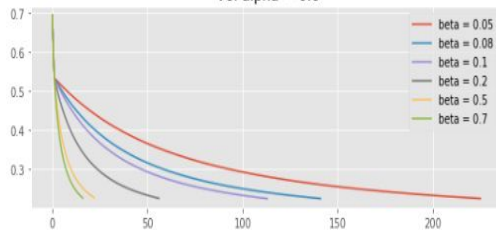
Với alpha = 0.5



Với alpha = 0.7



Với alpha = 0.6



voi alpha 0.5

Nếu beta = 0.05 vòng lặp là: 226 thời gian chạy là 0.3820149399998627 cost 0.2241400763830416
Nếu beta = 0.08 vòng lặp là: 142 thời gian chạy là 0.24680640900078288 cost 0.22401471675870943
Nếu beta = 0.1 cần số vòng lặp là: 114 thời gian chạy là 0.19327996799984248 cost 0.22393105546735825
Nếu beta = 0.2 cần số vòng lặp là: 57 thời gian chạy là 0.1018854289995943 cost 0.22467910504283914
Nếu beta = 0.5 cần số vòng lặp là: 23 thời gian chạy là 0.044548265999765135 cost 0.22639407839145095
Nếu beta = 0.7 cần số vòng lặp là: 17 thời gian chạy là 0.030283439000413637 cost 0.225471771606577

voi alpha 0.6

Nếu beta = 0.05 cần số vòng lặp là: 226 thời gian chạy là 0.3882792120002705 cost 0.2241400763830416
Nếu beta = 0.08 cần số vòng lặp là: 142 thời gian chạy là 0.24908627299919317 cost 0.22401471675870943
Nếu beta = 0.1 cần số vòng lặp là: 114 thời gian chạy là 0.19445511100093427 cost 0.22393105546735825
Nếu beta = 0.2 cần số vòng lặp là: 57 thời gian chạy là 0.1023762530003296 cost 0.22467910504283914
Nếu beta = 0.5 cần số vòng lặp là: 23 thời gian chạy là 0.041438378999373526 cost 0.22639407839145095
Nếu beta = 0.7 cần số vòng lặp là: 17 thời gian chạy là 0.029899610999564175 cost 0.225471771606577

voi alpha 0.7

Nếu beta = 0.05 cần số vòng lặp là: 226 thời gian chạy là 0.4212430640000093 cost 0.2241400763830416
Nếu beta = 0.08 cần số vòng lặp là: 142 thời gian chạy là 0.24767708299987135 cost 0.22401471675870943
Nếu beta = 0.1 cần số vòng lặp là: 114 thời gian chạy là 0.19990828700065322 cost 0.22393105546735825
Nếu beta = 0.2 cần số vòng lặp là: 57 thời gian chạy là 0.10483840100096131 cost 0.22467910504283914
Nếu beta = 0.5 cần số vòng lặp là: 23 thời gian chạy là 0.03876720900007058 cost 0.22639407839145095
Nếu beta = 0.7 cần số vòng lặp là: 17 thời gian chạy là 0.028690016999462387 cost 0.225471771606577

2. CÁC THUẬT TOÁN

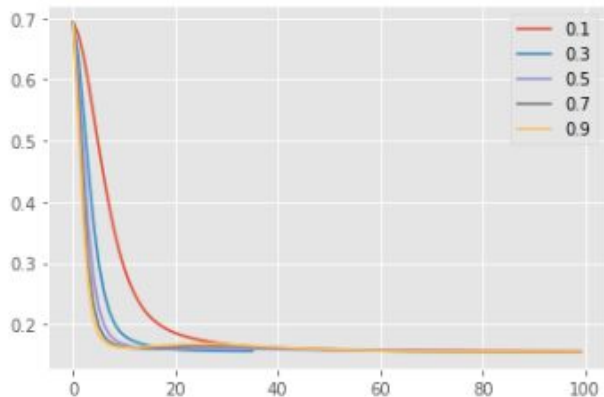
Dùng NAG và NAG backtracking

Dùng NAG với các hệ số eta khác nhau

Công thức cập nhật của NAG được cho như sau:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

voi step_size thời gian chạy là 0.1 0.0672651559998485
so vong lap can la 99
voi step_size thời gian chạy là 0.3 0.023309418999815534
so vong lap can la 35
voi step_size thời gian chạy là 0.5 0.06838478899999245
so vong lap can la 99
voi step_size thời gian chạy là 0.7 0.07717308200017214
so vong lap can la 99
voi step_size thời gian chạy là 0.9 0.07151751899982628
so vong lap can la 99

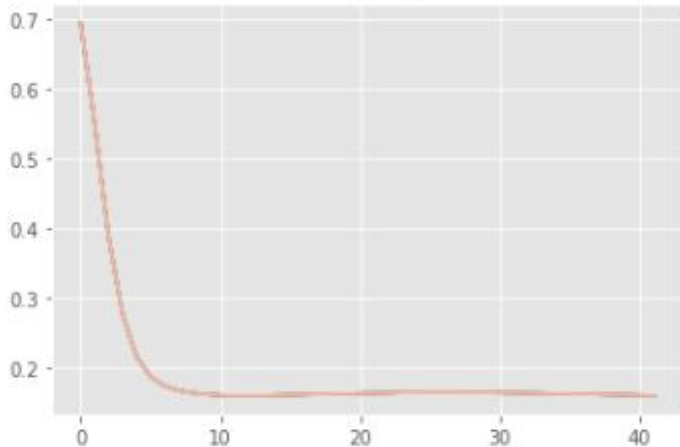


```
# check với NAG
steplist=[0.1,0.3,0.5,0.7,0.9]
for step in(steplist):
    start = timeit.default_timer()
    pars1,iterts1,cost1= GD_NAG(X,y,step,0.9,100)
    stop = timeit.default_timer()
    print('voi step_size',"thời gian chạy là", step, stop - start,"\n" "so vong lap can la",iterts1)
    label=("step size"+ str(step))
    plt.plot(np.arange(len(cost1)),cost1,label= label)
    plt.legend(["0.1","0.3","0.5","0.7","0.9"])]
```

2. CÁC THUẬT TOÁN

Dùng NAG và NAG backtracking

→ Nếu beta = 0.05 cần số vòng lặp là: 41
Nếu beta = 0.08 cần số vòng lặp là: 41
Nếu beta = 0.1 cần số vòng lặp là: 41
Nếu beta = 0.2 cần số vòng lặp là: 41
Nếu beta = 0.5 cần số vòng lặp là: 41
Nếu beta = 0.7 cần số vòng lặp là: 41
Nếu beta = 0.9 cần số vòng lặp là: 41



Backtracking under with acceleration in different ways.

Simple approach: fix $\beta < 1$, $t_0 = 1$. At iteration k , start with $t = t_{k-1}$, and while

$$g(x^+) > g(v) + \nabla g(v)^T(x^+ - v) + \frac{1}{2t}\|x^+ - v\|_2^2$$

shrink $t = \beta t$, and let $x^+ = \text{prox}_{th}(v - t\nabla g(v))$. Else keep x^+ .

2. CÁC THUẬT TOÁN

2.1. Dùng NAG và NAG backtracking

```
def GD_NAG_backtracking(X, y, eta, gamma, beta, T):
    w = np.zeros(X.shape[1])
    cost_store = [cost(X, y, w)]
    v = np.zeros_like(w)
    for it in range(1, T):
        learning_rate = 1
        v = gamma * v + eta * gradient(X, y, w - gamma * v)
        c = cost(X, y, w)
        g = gradient(X, y, w)
        while cost(X, y, w - learning_rate * v) > c - learning_rate * np.dot(g.T, v) + learning_rate * (np.linalg.norm(v) ** 2) / 2:
            learning_rate *= beta
        w = w - learning_rate * v
        cost_store.append(cost(X, y, w))
        if np.linalg.norm(gradient(X, y, w)) / len(w) < 1e-3:
            break
    return cost_store, it

# chạy hàm trên

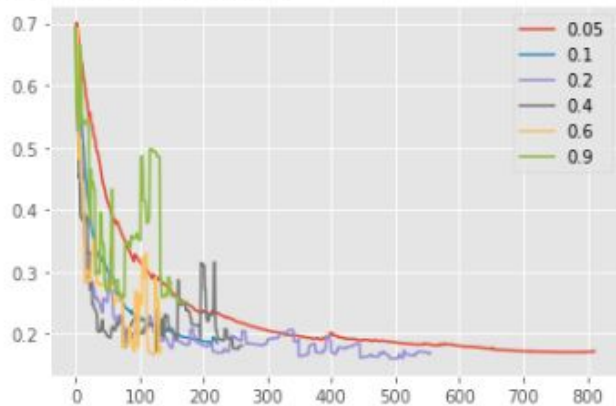
num_iters_bt = 500
beta_ = np.array([0.05, 0.08, 0.1, 0.2, 0.5, 0.7, 0.9])

for beta in beta_:
    cost1, maxIters = GD_NAG_backtracking(X, y, 0.8, 0.9, beta, 500)
    print("Nếu beta =", beta, "cần số vòng lặp là:", maxIters)
    plt.plot(np.arange(len(cost1)), cost1)
```

2. CÁC THUẬT TOÁN

Dùng Schotastic Gradient Descent

voi step_size thời gian chạy là 0.05 0.35941030499998305
so vong lap can la 811
voi step_size thời gian chạy là 0.1 0.09425065300001734
so vong lap can la 221
voi step_size thời gian chạy là 0.2 0.25584118200004013
so vong lap can la 554
voi step_size thời gian chạy là 0.4 0.11401532600029896
so vong lap can la 260
voi step_size thời gian chạy là 0.6 0.05841800100006367
so vong lap can la 134
voi step_size thời gian chạy là 0.9 0.07913718500003597
so vong lap can la 174



```
def sgd(X, y, lr, epochs):  
    theta_0 = np.zeros(D)  
    theta_list = [theta_0]  
    loss_0 = cost(X, y, theta_0)  
    loss_list = [loss_0]  
    iters = 0  
    for t in range(epochs):  
        rand_index = np.random.permutation(N)  
        for i in rand_index:  
            xi = X[i, :][np.newaxis]  
            yi = y[i][np.newaxis]  
            grad = gradient(xi, yi, theta_list[-1])  
            theta_new = theta_list[-1] - lr * grad  
            loss_new = cost(X, y, theta_new)  
            if np.abs(loss_new - loss_list[-1]) < 1e-6:  
                return theta_list, loss_list, iters, t+1  
            iters += 1  
            loss_list.append(loss_new)  
            theta_list.append(theta_new)  
    return theta_list, loss_list, iters, t+1
```

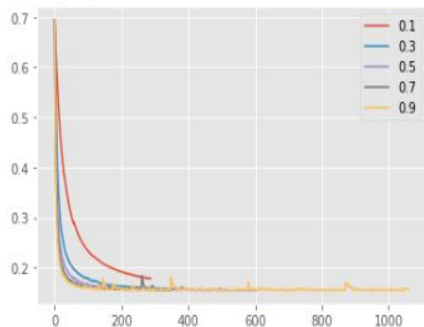
```
steplist=[0.05,0.1,0.2,0.4,0.6,0.9]  
for step in(steplist):  
    start = timeit.default_timer()  
    pars1,cost1,iterts1,tx= sgd(X,y,step,100)  
    stop = timeit.default_timer()  
    print('voi step_size',"thời gian chạy là", step, stop - start,"\n" "so vong lap can la",iterts1)  
    label=("step size"+ str(step))  
    plt.plot(np.arange(len(cost1)),cost1,label= label)  
    plt.legend(["0.05","0.1","0.2",0.4,0.6, 0.9])
```


2. CÁC THUẬT TOÁN

Dùng Mini_Batch Gradient Descent

```
steplist=[0.1,0.3,0.5,0.7,0.9]
for step in(steplist):
    start = timeit.default_timer()
    cost1, iterts1= mgd(X,y,step,100,30)
    stop = timeit.default_timer()
    print('voi step_size', step,"thời gian chạy là", stop - start,"\n" "so vong lap can la",iterts1)
    label=("step size"+ str(step))
    plt.plot(np.arange(len(cost1)),cost1,label= label)
    plt.legend(["0.1","0.3","0.5","0.7","0.9"])
```

voi step_size 0.5 thời gian chạy là 0.9609526600002027
so vong lap can la 602
voi step_size 0.7 thời gian chạy là 0.5599332110000432
so vong lap can la 351
voi step_size 0.9 thời gian chạy là 1.6838938850000886
so vong lap can la 1060

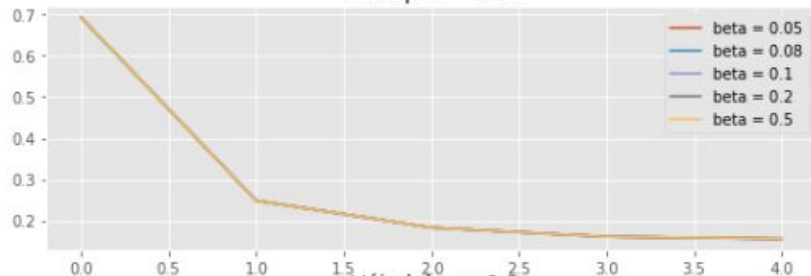


```
def mgd(X, y, alpha, T, batch_size=32):
    cost_store = []
    iter = 1
    theta = np.zeros((D))
    cost_store.append(cost(X, y,theta))
    for t in range(1, T):
        X, y = shuffle(X, y)
        # print(t)
        for i in range(0, N, batch_size):
            X_i = X[i:i + batch_size]
            y_i = y[i:i + batch_size]
            grad = gradient(X_i, y_i,theta)
            new_theta = theta - alpha * grad
            cost_store.append(cost(X, y,new_theta))
            if np.abs(cost(X, y, new_theta) - cost(X, y,theta)) < 1e-6:
                return cost_store, iter
            theta = new_theta
            iter += 1
    return cost_store, iter - 1
```

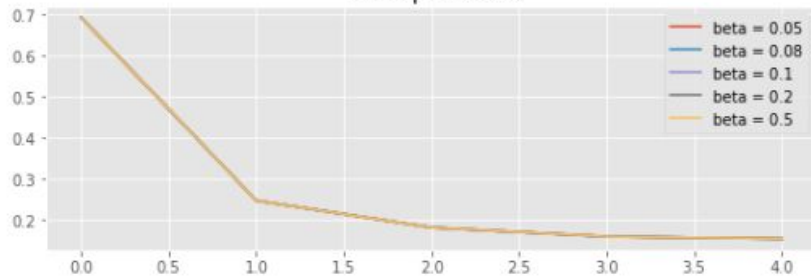
2. CÁC THUẬT TOÁN

Dùng Phương pháp Newton(và cả Backtracking)

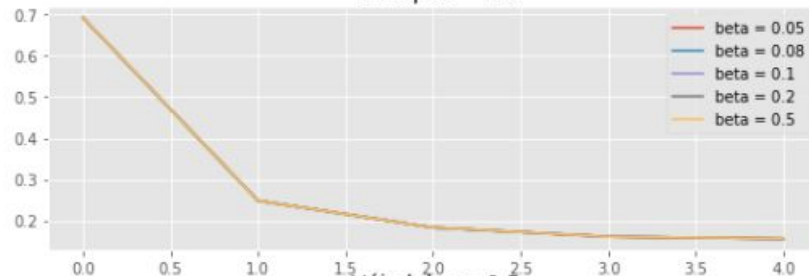
Với $\alpha = 0.05$



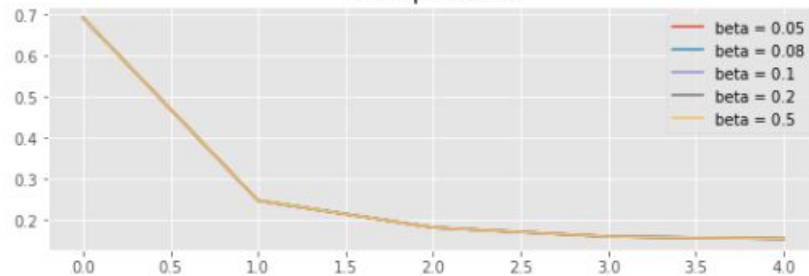
Với $\alpha = 0.3$




Với $\alpha = 0.1$



Với $\alpha = 0.5$





Newton's method trong bài toán tìm local minimum

Áp dụng phương pháp này cho việc giải phương trình $f'(x) = 0$ ta có:

$$x_{t+1} = x_t - (f''(x_t))^{-1} f'(x_t)$$

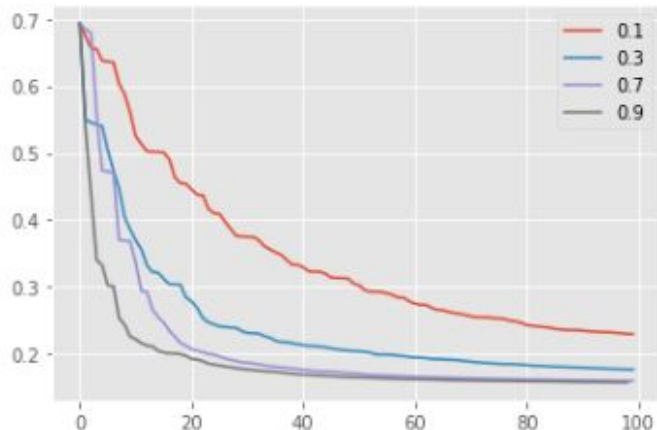
Và trong không gian nhiều chiều với θ là biến:

$$\theta = \theta - \mathbf{H}(J(\theta))^{-1} \nabla_{\theta} J(\theta)$$

2. CÁC THUẬT TOÁN

Randomized Coordinate Gradient Descent

```
for step in([0.1,0.3,0.7,0.9]):  
    i,o,p,u=Rand_CD(X,y,step,100)  
    plt.plot(np.arange(len(o)),o)  
    plt.legend([0.1,0.3,0.7,0.9])
```



```
def Rand_CD(X, y, lr, T):  
    D = X.shape[1]  
    w = np.zeros(D)  
    cost_store = []  
    iter = 0  
    for t in range(0, T):  
        cost_store.append(cost(X, y,w))  
        for i in range(0, D):  
            i = np.random.randint(0, D)  
            w[i] = w[i] - lr * gradient(X, y,w)[i]  
            iter += 1  
        if np.linalg.norm(gradient(X, y,w)) / D < 1e-3:  
            break  
    return w, cost_store, iter, t+1
```

2. CÁC THUẬT TOÁN

Randomized Coordinate Gradient Descent

Randomized Coordinate Gradient Descent

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
    obj_val = obj(w)
    for i in range(0, dim):
        i = np.random.randint(0, dim)
        w[i] = w[i] - 1/L * grad(w)[i]
```

2. CÁC THUẬT TOÁN

Cyclic Coordinate Gradient Descent

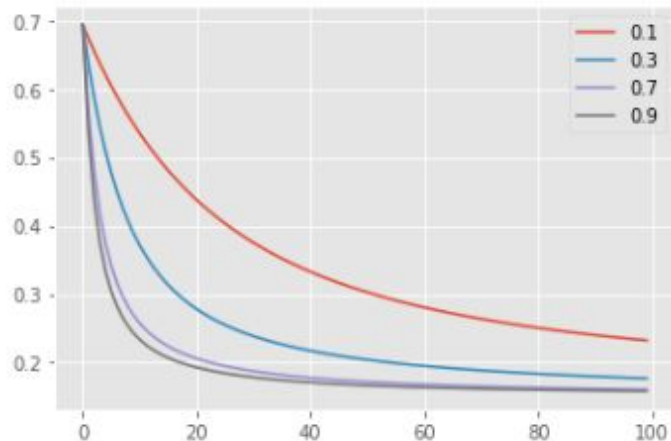
Cyclic Coordinate Gradient Descent

```
w = np.matrix([0.0]*dim).T
for t in range(0, max_iter):
    obj_val = obj(w)
    for i in range(0,dim):
        w[i] = w[i] - 1/L * grad(w)[i]
```

2. CÁC THUẬT TOÁN

Cyclic Coordinate Gradient Descent

```
for step in([0.1,0.3,0.7,0.9]):  
    i,o,p,u= Cyclic_CGD(X,y,step,100)  
    plt.plot(np.arange(len(o)),o)  
    plt.legend([0.1,0.3,0.7,0.9]) |
```



```
def Cyclic_CGD(X, y, lr, T):  
    D = X.shape[1]  
    w = np.zeros(D)  
    cost_store = []  
    iter = 0  
    for t in range(0, T):  
        cost_store.append(cost(X, y,w))  
        for i in range(0, D):  
            w[i] = w[i] - lr * gradient(X, y,w)[i]  
            iter += 1  
        if np.linalg.norm(gradient(X, y,w)) / D < 1e-3:  
            break  
    return w, cost_store, iter, t
```

2. CÁC THUẬT TOÁN

Fast proximal gradient methods (FISTA)

FISTA

$$w_{t+1} = v_t - \frac{1}{L} \nabla f(v_t)$$
$$v_{t+1} = w_{t+1} + \frac{a_t - 1}{a_{t+1}} (w_{t+1} - w_t)$$

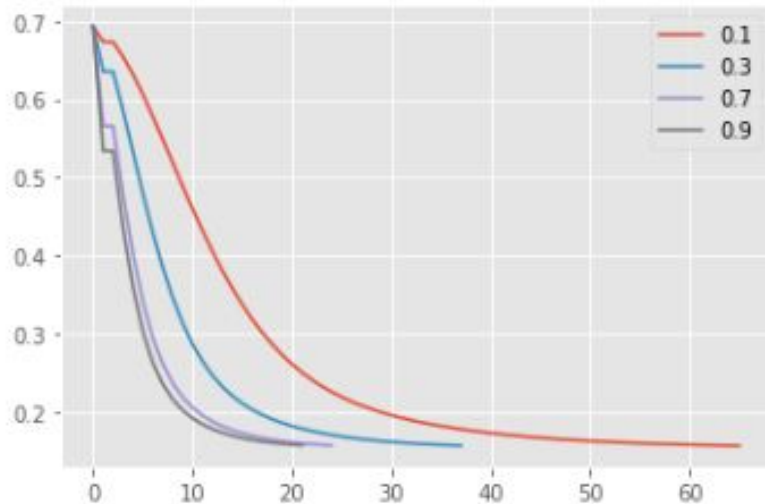
where $a_0 = 0$, $a_{t+1} = \frac{1 + \sqrt{1 + 4a_t^2}}{2}$.

```
w = np.matrix([0.0]*dim).T
v = w
a = 0.0
for t in range(0, max_iter):
    obj_val = obj(w)
    w_prev = w
    w = v - 1/L*grad(v)
    a_prev = a
    a = (1 + np.sqrt(1 + 4 * a_prev**2))/2
    v = w + (a_prev - 1) / a * (w - w_prev)
```

2. CÁC THUẬT TOÁN

Fast proximal gradient methods (FISTA)

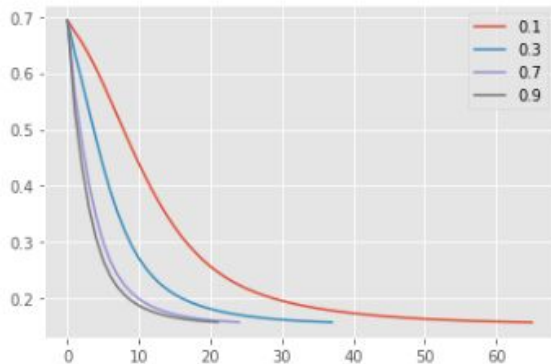
```
for step in([0.1,0.3,0.7,0.9]):  
    i,o,p= FISTA(X,y,step,100)  
    plt.plot(np.arange(len(o)),o)  
    plt.legend([0.1,0.3,0.7,0.9])
```



2. CÁC THUẬT TOÁN

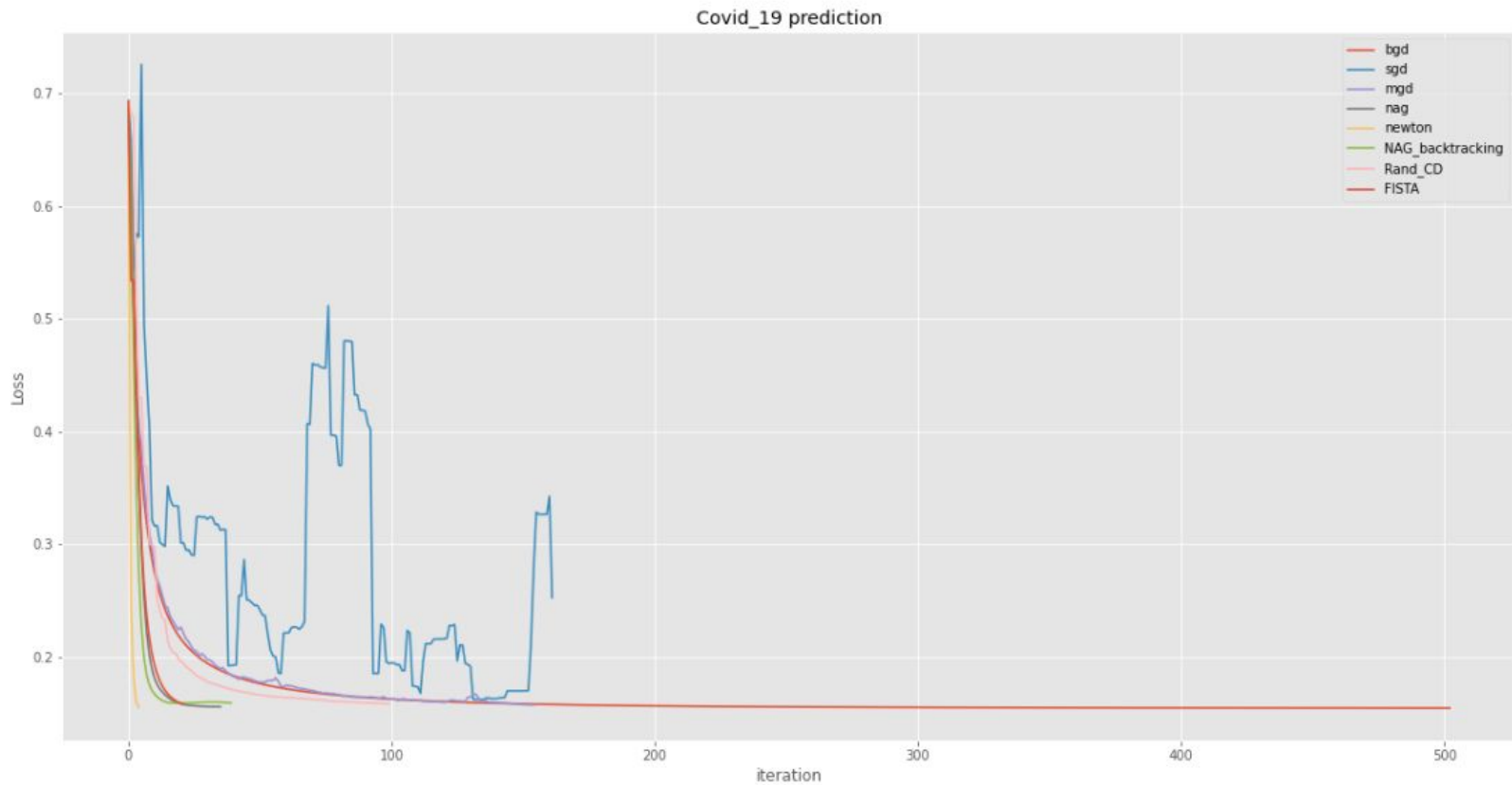
Accelerated Gradient Descent

```
for step in([0.1,0.3,0.7,0.9]):  
    i,o,p= accelerated_gd(X,y,step,100)  
    plt.plot(np.arange(len(o)),o)  
    plt.legend([0.1,0.3,0.7,0.9])
```



```
def accelerated_gd(X, y, lr, iterations):  
    N = X.shape[0]  
    D = X.shape[1]  
    loss_list = []  
    theta = np.zeros((D))  
    for t in range(iterations):  
        loss_list.append(cost(X, y, theta))  
        if t == 0:  
            grad = gradient(X, y, theta)  
            last_theta = theta  
            theta = theta - lr * grad  
        else:  
            v = theta + (t-1)/(t+2) * (theta - last_theta)  
            last_theta = theta  
            grad = gradient(X, y, v)  
            theta = v - lr * grad  
  
        if np.linalg.norm(grad)/D < 1e-3:  
            break  
  
    return theta, loss_list, t
```


2. CÁC THUẬT TOÁN



2. CÁC THUẬT TOÁN



bgd Time: 0.3219629030008946
so vong lap can la 503
loss = 0.15455101030361423

sgd Time: 0.06868963699889719
so vong lap can la 161
loss = 0.2525758778288807

mgd Time: 0.26053010600116977
so vong lap can la 154
loss = 0.15736702038011294

nag Time: 0.024216625999542885
so vong lap can la 36
loss = 0.15579443976821875

newton Time: 0.03469031900021946
so vong lap can la 4
loss = 0.15501848696289658

NAG_backtracking time: 0.08280298600038805
so vong lap can la 39
loss = 0.15922738433357625

Rand_CD time: 0.2995610529997066
so vong lap can la 800
loss = 0.15845715174985092

FISTA time: 0.020757426998898154
so vong lap can la 21
loss = 0.15788554141211278

3. Nhận xét về các thuật toán

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
logisticRegr.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

```
predictions = logisticRegr.predict(X_test)
score = logisticRegr.score(X_test, y_test)
print(score)
```

```
0.9408284023668639
```

Nếu dùng thư viện, accuracy là 94%

3.NHẬN XÉT VỀ CÁC THUẬT TOÁN

```
| y_predict_bgd = predict(X_test, pars_fixed[-1])  
print('bgd accuracy: ', np.sum(y_predict_bgd == y_test)/len(X_test))
```

bgd accuracy: 0.9585798816568047

```
y_predict_sgd = predict(X_test, pars1_sgd[-1])  
print('sgd accuracy: ', np.sum(y_predict_sgd == y_test)/len(X_test))
```

sgd accuracy: 0.8994082840236687

```
| y_predict_nag = predict(X_test, w_mm)  
print('nag accuracy: ', np.sum(y_predict_nag == y_test)/len(X_test))  
print(confusion_matrix(y_test, y_predict_nag))
```

nag accuracy: 0.9585798816568047

```
[[155  0]  
 [ 7  7]]
```

```
from sklearn.metrics import confusion_matrix  
y_predict_newton = predict(X_test, w_newton)  
print('Newton accuracy: ', np.sum(y_predict_newton == y_test)/len(X_test))  
print(confusion_matrix(y_test, y_predict_newton))
```

Newton accuracy: 0.9289940828402367

Test thử với vài mô hình cho độ chính xác tương tự

3. Nhận xét về các thuật toán



Nhận xét:

- Phương pháp NAG khắc phục việc nghiệm của GD rơi vào một điểm local minimum không mong muốn. nếu dữ liệu lớn hơn thì người ta dùng phương pháp momentum gradient descent.
- BGD vẫn luôn rất chậm nhưng ổn định, nhưng không tránh khỏi local minimum
- Khi áp dụng Newton's method cho bài toán tối ưu trong không gian nhiều chiều, chúng ta cần tính nghịch đảo của Hessian matrix. Khi số chiều và số điểm dữ liệu lớn, đạo hàm bậc hai của hàm mất mát sẽ là một ma trận rất lớn, ảnh hưởng tới cả memory và tốc độ tính toán của hệ thống nên không khuyến khích dùng khi đó. Tiếp nữa là nó ảnh hưởng rất lớn bởi giá trị tham số ban đầu chọn.
- Người ra thường dùng mini batch để mô hình ổn định hơn so với SGD