In [154]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

Import and cleaning data

In [155]:

```
raw = pd.read_csv("voice.csv")
```

In [156]:

```
raw.head()
```

Out[156]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	ma
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.059781	0.084279	0.015702	0.275
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.066009	0.107937	0.015826	0.250
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.077316	0.098706	0.015656	0.271
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.151228	0.088965	0.017798	0.250
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.135120	0.106398	0.016931	0.266

5 rows × 21 columns

In [157]:
```
raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object 
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

Xét mối quan hệ các biến với biến phụ thuộc

In [158]:
```
raw['label'], x =pd.factorize(raw['label'])
```

In [159]:
```
corr = raw.corr()
corr
```

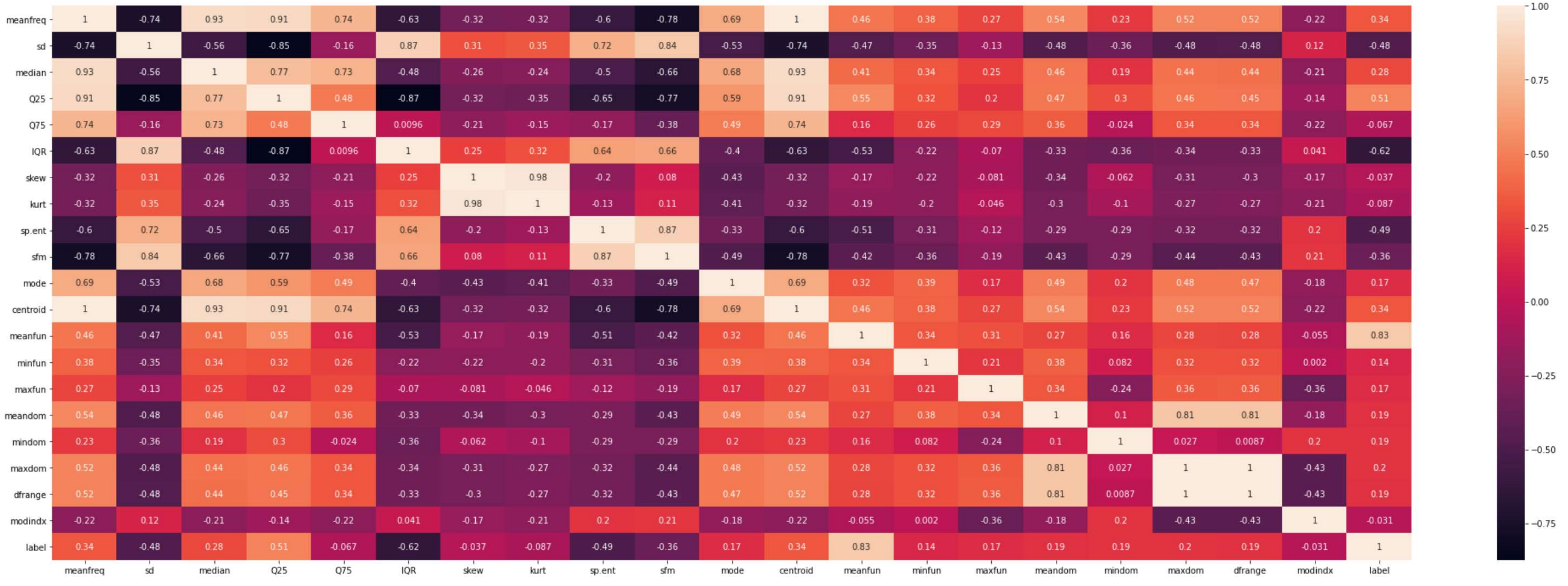
Out[159]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	m
meanfreq	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036	-0.601203	-0.784332	...	1.000000	0.460844	0.38
sd	-0.739039	1.000000	-0.562603	-0.846931	-0.161076	0.874660	0.314597	0.346241	0.716620	0.838086	...	-0.739039	-0.466281	-0.34
median	0.925445	-0.562603	1.000000	0.774922	0.731849	-0.477352	-0.257407	-0.243382	-0.502005	-0.661690	...	0.925445	0.414909	0.33
Q25	0.911416	-0.846931	0.774922	1.000000	0.477140	-0.874189	-0.319475	-0.350182	-0.648126	-0.766875	...	0.911416	0.545035	0.32
Q75	0.740997	-0.161076	0.731849	0.477140	1.000000	0.009636	-0.206339	-0.148881	-0.174905	-0.378198	...	0.740997	0.155091	0.25
IQR	-0.627605	0.874660	-0.477352	-0.874189	0.009636	1.000000	0.249497	0.316185	0.640813	0.663601	...	-0.627605	-0.534462	-0.22
skew	-0.322327	0.314597	-0.257407	-0.319475	-0.206339	0.249497	1.000000	0.977020	-0.195459	0.079694	...	-0.322327	-0.167668	-0.21
kurt	-0.316036	0.346241	-0.243382	-0.350182	-0.148881	0.316185	0.977020	1.000000	-0.127644	0.109884	...	-0.316036	-0.194560	-0.20
sp.ent	-0.601203	0.716620	-0.502005	-0.648126	-0.174905	0.640813	-0.195459	-0.127644	1.000000	0.866411	...	-0.601203	-0.513194	-0.30
sfm	-0.784332	0.838086	-0.661690	-0.766875	-0.378198	0.663601	0.079694	0.109884	0.866411	1.000000	...	-0.784332	-0.421066	-0.36
mode	0.687715	-0.529150	0.677433	0.591277	0.486857	-0.403764	-0.434859	-0.406722	-0.325298	-0.485913	...	0.687715	0.324771	0.38
centroid	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036	-0.601203	-0.784332	...	1.000000	0.460844	0.38
meanfun	0.460844	-0.466281	0.414909	0.545035	0.155091	-0.534462	-0.167668	-0.194560	-0.513194	-0.421066	...	0.460844	1.000000	0.33
minfun	0.383937	-0.345609	0.337602	0.320994	0.258002	-0.222680	-0.216954	-0.203201	-0.305826	-0.362100	...	0.383937	0.339387	1.00
maxfun	0.274004	-0.129662	0.251328	0.199841	0.285584	-0.069588	-0.080861	-0.045667	-0.120738	-0.192369	...	0.274004	0.311950	0.21
meandom	0.536666	-0.482726	0.455943	0.467403	0.359181	-0.333362	-0.336848	-0.303234	-0.293562	-0.428442	...	0.536666	0.270840	0.37
mindom	0.229261	-0.357667	0.191169	0.302255	-0.023750	-0.357037	-0.061608	-0.103313	-0.294869	-0.289593	...	0.229261	0.162163	0.08
maxdom	0.519528	-0.482278	0.438919	0.459683	0.335114	-0.337877	-0.305651	-0.274500	-0.324253	-0.436649	...	0.519528	0.277982	0.31
dfrange	0.515570	-0.475999	0.435621	0.454394	0.335648	-0.331563	-0.304640	-0.272729	-0.319054	-0.431580	...	0.515570	0.275154	0.31
modindx	-0.216979	0.122660	-0.213298	-0.141377	-0.216475	0.041252	-0.169325	-0.205539	0.198074	0.211477	...	-0.216979	-0.054858	0.00
label	0.337415	-0.479539	0.283919	0.511455	-0.066906	-0.618916	-0.036627	-0.087195	-0.490552	-0.357499	...	0.337415	0.833921	0.13

21 rows × 21 columns

In [160]:
```python
plt.subplots(figsize=(36,12))
sns.heatmap(corr, annot=True)
```

Out[160]: <matplotlib.axes._subplots.AxesSubplot at 0x14b27738700>



In [161]:
```python
corr['label'].sort_values()
```

Out[161]:
```
IQR         -0.618916
sp.ent      -0.490552
sd          -0.479539
sfm         -0.357499
kurt        -0.087195
Q75         -0.066906
skew        -0.036627
modindx     -0.030801
minfun       0.136692
maxfun       0.166461
mode         0.171775
meandom      0.191067
dfrange      0.192213
mindom       0.194974
maxdom       0.195657
median       0.283919
meanfreq     0.337415
centroid     0.337415
Q25          0.511455
meanfun      0.833921
label        1.000000
Name: label, dtype: float64
```

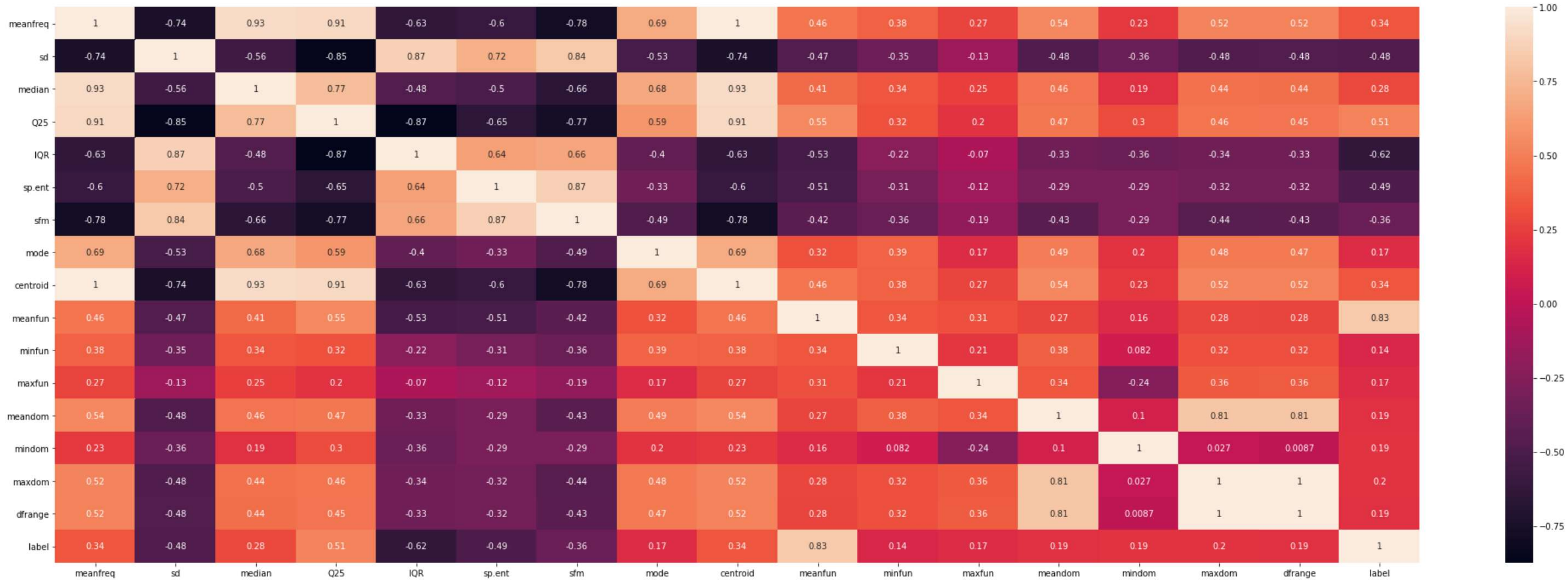
In [162]:
```python
data = raw.drop(["kurt","Q75","skew","modindx"],axis = 1)
data.head()
```

Out[162]:

	meanfreq	sd	median	Q25	IQR	sp.ent	sfm	mode	centroid	meanfun	minfun	maxfun	meandom	mindom	m
0	0.059781	0.064241	0.032027	0.015071	0.075122	0.893369	0.491918	0.000000	0.059781	0.084279	0.015702	0.275862	0.007812	0.007812	0
1	0.066009	0.067310	0.040229	0.019414	0.073252	0.892193	0.513724	0.000000	0.066009	0.107937	0.015826	0.250000	0.009014	0.007812	0
2	0.077316	0.083829	0.036718	0.008701	0.123207	0.846389	0.478905	0.000000	0.077316	0.098706	0.015656	0.271186	0.007990	0.007812	0
3	0.151228	0.072111	0.158011	0.096582	0.111374	0.963322	0.727232	0.083878	0.151228	0.088965	0.017798	0.250000	0.201497	0.007812	0
4	0.135120	0.079146	0.124656	0.078720	0.127325	0.971955	0.783568	0.104261	0.135120	0.106398	0.016931	0.266667	0.712812	0.007812	5

```
In [163]: corr = data.corr()
          plt.subplots(figsize=(36,12))
          sns.heatmap(corr, annot=True)
```

```
Out[163]: <matplotlib.axes._subplots.AxesSubplot at 0x14b201be1f0>
```



Xét sự mối quan hệ các biến: median ~ meanfreq + Q25 + mode + centroid

In [164]:
```python
median_data = data[["median","meanfreq", "Q25","mode","centroid"]]
corr1 = median_data.corr()
plt.subplots(figsize=(12,6))
sns.heatmap(corr1, annot=True)
```

Out[164]: <matplotlib.axes._subplots.AxesSubplot at 0x14b26d45ac0>

Sử dụng PCA giảm chiều dữ liệu

In [165]:
```python
# Importing standardscalar module
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

# fitting
scalar.fit(median_data)
scaled_data = scalar.transform(median_data)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components = 1)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

print(x_pca.shape)
print(sum(pca.explained_variance_ratio_))
```

```
(3168, 1)
0.8528680035258814
```

In [166]:
```python
data = data.drop(["median","meanfreq", "Q25","mode","centroid"],axis = 1)
data['_median'] = x_pca
```

In [167]:
```python
data.head()
```

Out[167]:

	sd	IQR	sp.ent	sfm	meanfun	minfun	maxfun	meandom	mindom	maxdom	dfrange	label	_median
0	0.064241	0.075122	0.893369	0.491918	0.084279	0.015702	0.275862	0.007812	0.007812	0.007812	0.000000	0	7.733128
1	0.067310	0.073252	0.892193	0.513724	0.107937	0.015826	0.250000	0.009014	0.007812	0.054688	0.046875	0	7.392225
2	0.083829	0.123207	0.846389	0.478905	0.098706	0.015656	0.271186	0.007990	0.007812	0.015625	0.007812	0	7.172246
3	0.072111	0.111374	0.963322	0.727232	0.088965	0.017798	0.250000	0.201497	0.007812	0.562500	0.554688	0	2.088342
4	0.079146	0.127325	0.971955	0.783568	0.106398	0.016931	0.266667	0.712812	0.007812	5.484375	5.476562	0	3.082777

In [168]:
```
y = data["label"].values
X = data.drop(["label"],axis = 1)
```

In [169]:
```
from sklearn.preprocessing import Normalizer
transformer = Normalizer().fit(X)
X_nor =transformer.transform(X)
```

In [170]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X_nor,y,test_size=0.2,random_state=42)
x_train=x_train.T
x_test=x_test.T
y_train=y_train.T
y_test=y_test.T
```

Batch Gradient descent method

In [171]:
```
# Khởi tạo W0, b0 ban đầu.
def initialize_weights_and_bias(dimension):
    w = np.full((dimension,1),0)
    b = 0.0
    return w,b
```

In [172]:
```
# Tính giá trị thực của Y : log(y/1-y) = z => y = 1/(1+e^-z)
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

In [173]:
```
#Gradient descent
def batch_gradient_descent(w,b,x_train,y_train):
    #Phương trình linear Z = wX + b
#     b = 0
    z = np.dot(w.T,x_train)+b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    # tính LSQ = ((y - ypredict)^2)/n
    cost = (np.sum(loss))/x_train.shape[1]
    ## Độ lệch của w
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1]
    #Độ lệch của b
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]
    gradients = {"derivative_weight":derivative_weight,"derivative_bias":derivative_bias}
    return cost,gradients
```

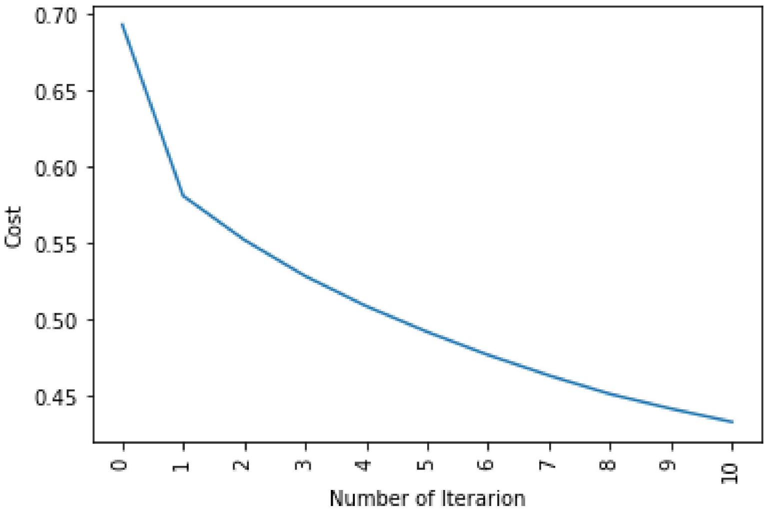
In [174]:
```
def update(w,b,x_train,y_train,learning_rate,time_interation):
    cost_list=[]
    cost_list2=[]
    index=[]
    start = time.time()
    point = 0
    interaction = 0
    while (time.time()-start<time_interation):
        interaction+=1
        cost,gradients = batch_gradient_descent(w,b,x_train,y_train)
        cost_list.append(cost)
        ##TÍnh lại trọng số w
        w = w - learning_rate*gradients["derivative_weight"]
        #Tính lại trọng số b
        b = b - learning_rate*gradients["derivative_bias"]
        #Lưu kết quả vòng lặp
        if (time.time() - start) > point:
            cost_list2.append(cost)
            index.append(point)
            print ("Cost after %i s: %f" %(point,cost))
            point += 1
    parameters={"weight":w,"bias":b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    global index_GD
    global cost_list_GD
    index_GD = index
    cost_list_GD = cost_list2
    return parameters,gradients,index,cost_list,interaction
```

In [175]:
```
def predict(w,b,x_test):
#     b = 0
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_pre = np.zeros((1,x_test.shape[1]))

    for i in range(z.shape[1]):
        if z[0,i]<=0.5:
            Y_pre[0,i] = 0
        else:
            Y_pre[0,i]=1
    return Y_pre
```

In [176]:
```
def logistic_regression(x_train,x_test,y_train,y_test,learning_rate,time_interation):
    ## Cỡ mẫu
    dimension = x_train.shape[0]
    #Khởi tạo w0,b0
    w,b = initialize_weights_and_bias(dimension)
    parameters,gradients,index,cost_list2,interaction = update(w,b,x_train,y_train,learning_rate,time_interation)
    y_pre_test = predict(parameters["weight"],parameters["bias"],x_test)
    print("test accuracy:{}%".format(100-np.mean(np.abs(y_pre_test-y_test)*100)))
    print("W = ",parameters["weight"])
    print("b = ",parameters["bias"])
    print("Number of interaction: ",interaction)
```

In [177]:
```
logistic_regression(x_train,x_test,y_train,y_test,learning_rate=2,time_interation=10)
```

```
Cost after 0 s: 0.693147
Cost after 1 s: 0.580773
Cost after 2 s: 0.552024
Cost after 3 s: 0.528222
Cost after 4 s: 0.508560
Cost after 5 s: 0.491644
Cost after 6 s: 0.476470
Cost after 7 s: 0.463023
Cost after 8 s: 0.450875
Cost after 9 s: 0.441229
Cost after 10 s: 0.432558
```



```
test accuracy:89.58990536277602%
W =  [[ -4.35543024]
 [-46.24764455]
 [-18.2546774 ]
 [  7.44409604]
 [ 86.23514017]
 [ -2.30439771]
 [ 12.92968868]
 [ -0.64661377]
 [  8.03662577]
 [  2.19022024]
 [ -5.84640553]
 [ -1.84894646]]
b =  2.7473613961409336
Number of interaction:  27550
```

Stochatic Gradient descent

In [178]:

```
x_train1 = np.vstack((np.ones([1,x_train.shape[1]]),x_train))
x_test1 = np.vstack((np.ones([1,x_test.shape[1]]),x_test))
y_train = y_train.reshape((1,-1))
```

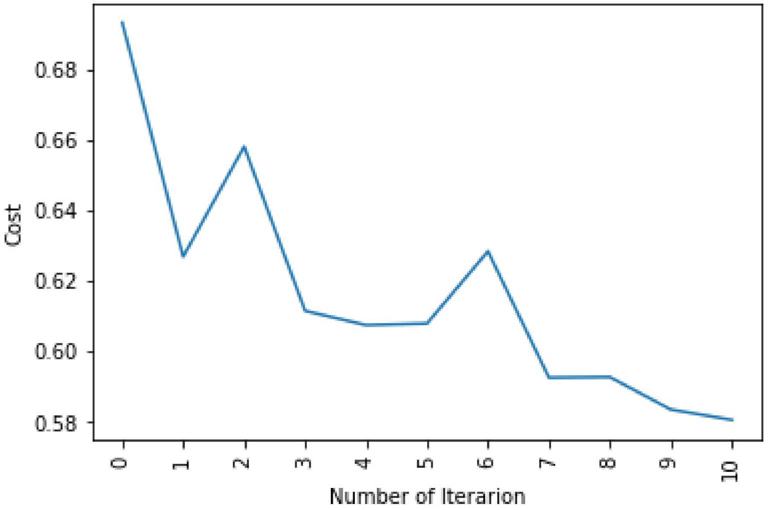
```
In [179]: def update_stochastic(w_init,x_train1,y_train,learning_rate,time_interation):
    cost_list=[]
    cost_list2=[]
    index=[]
    w = w_init
    i = 1
    point = 0
    start =time.time()
    interaction = 0
    while (time.time()-start)<time_interation:
        id_random = np.random.permutation(x_train1.shape[1])
        for j in id_random:
            interaction+=1
            z = np.dot(w.T,x_train1)
            y_head = sigmoid(z)
            loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
            cost = (np.sum(loss))/x_train1.shape[1]
            # Tính lại hệ số
            w = w - learning_rate/np.log(i+1)*(y_head[0,j] - y_train[0,j])*x_train1[:,j].reshape(-1,1)
            #Lưu kết quả vòng lặp
            if (time.time()-start)>point:
                cost_list2.append(cost)
                index.append(point)
                print ("Cost after %i s: %f" %(point, cost))
                point += 1
            i += 1
            if (time.time()-start)>time_interation:
                break
#             if i > num_iteration:
#                 break
    parameters = w
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    global index_SGD
    global cost_list_SGD
    index_SGD = index
    cost_list_SGD = cost_list2
    return parameters,interaction
```

In [180]:
```
def predict_stochastic(w,x_test1):
    z = sigmoid(np.dot(w.T,x_test1))
    Y_pre = np.zeros((1,x_test1.shape[1]))
    for i in range(z.shape[1]):
        if z[0,i]<=0.5:
            Y_pre[0,i] = 0
        else:
            Y_pre[0,i]=1

    return Y_pre
```

In [181]:
```
def SGD_test(x_train1,x_test1,y_train,y_test,learning_rate,time_interation):

    dimension = x_train1.shape[0]
    w_init,b = initialize_weights_and_bias(dimension)
    parameters,interaction = update_stochastic(w_init,x_train1,y_train,learning_rate,time_interation)
    y_pre_test = predict_stochastic(parameters,x_test1)
    print("test accuracy:{}%".format(100-np.mean(np.abs(y_pre_test-y_test)*100)))
    print("W = ",parameters)
    print("Number of interaction: ",interaction)
```

In [182]:
```
SGD_test(x_train1,x_test1,y_train,y_test,learning_rate=2,time_interation=10)
```

```
Cost after 0 s: 0.693147
Cost after 1 s: 0.626786
Cost after 2 s: 0.657998
Cost after 3 s: 0.611380
Cost after 4 s: 0.607351
Cost after 5 s: 0.607801
Cost after 6 s: 0.628260
Cost after 7 s: 0.592459
Cost after 8 s: 0.592604
Cost after 9 s: 0.583392
Cost after 10 s: 0.580459
```

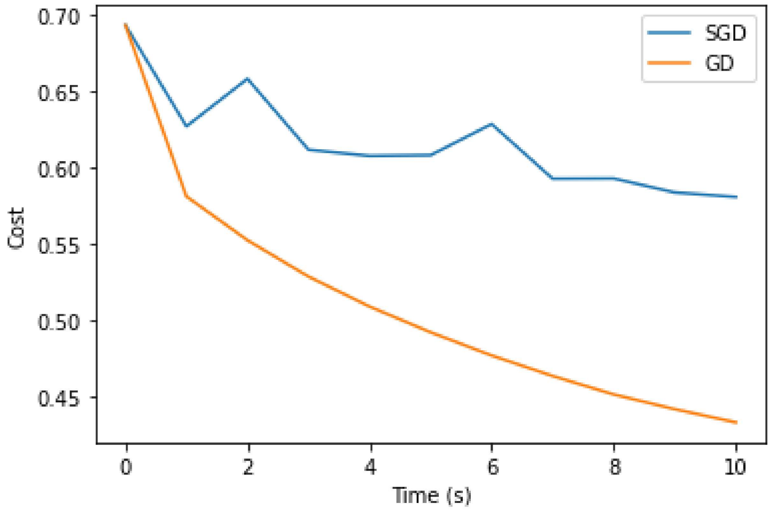



```
test accuracy:75.39432176656152%
W =  [[ 2.41808047]
 [-0.92452365]
 [-9.55071464]
 [-3.7805258 ]
 [ 1.92524441]
 [15.57688757]
 [-0.15123772]
 [ 2.70948395]
 [-1.49476672]
 [ 2.55534849]
```

```
 [-0.49632012]
 [-3.05166861]
 [-2.97458895]]
Number of interaction:  34334
```

In [183]:

```python
plt.plot(index_SGD,cost_list_SGD,label = "SGD")
plt.plot(index_GD,cost_list_GD,label = "GD")
plt.legend()
plt.xlabel("Time (s)")
plt.ylabel("Cost")
plt.show()
```



So sánh thuật toán với thư viện có sẵn trong sklearn

In [184]:

```python
import warnings
warnings.filterwarnings('ignore')
```

In [185]:
```
# chạy SGD, loss mặc định = 'hinge' => thuật toán SVM
schotasticGD = SGDClassifier(random_state=42)
```

In [186]:
```
%%time
schotasticGD.fit(x_train.T, y_train.T)
sgd = cross_val_score(schotasticGD, x_test.T, y_test.T, cv=50, scoring="accuracy")
print("accuracy = "+ str(sgd.mean()))
```

```
accuracy = 0.662051282051282
Wall time: 188 ms
```

In [187]:
```
# chạy BGD, loss mặc định = 'hinge' => thuật toán SVC
batchGD = SVC(random_state=42)
```

In [188]:
```
%%time
batchGD.fit(x_train.T, y_train.T)
svm = cross_val_score(batchGD, x_test.T, y_test.T, cv=50, scoring="accuracy")
print("accuracy = "+ str(svm.mean()))
```

```
accuracy = 0.7397435897435897
Wall time: 780 ms
```

Nhận xét:

1. Cùng với 1 khoảng thời gian thuật toán BGD cho ra độ chính xác cao hơn: ở đây trong cùng khoảng thời gian chạy là 10s, trong khi thuật toán SGD cho ra accuracy = 75% thì thuật toán BGD cho ra accuracy = 90 %

2. Quan sát đồ thị cost cuả 2 thuật toán có thể thấy thuật toán BGD có tính ổn định cao hơn thuật toán SGD

3. Với cùng khoảng thời gian là 10s ta thấy số vòng lặp của phương pháp BGD là 27550 trong khi số vòng lặp được thực hiện của phương pháp SGD là 34334. Do trong mỗi bước lặp, thuật toán BGD sẽ duyệt qua tất cả các phần tử của tập train trong khi thuật toán SGD chỉ duyệt qua 1 phần tử random trong đó do vậy về không gian bộ nhớ thuật toán SGD sẽ tối ưu hơn thuật toán BGD

4. Khi kiểm so sánh độ chính xác của thuật toán với thư viện có sẵn trong sklearn thấy độ chính xác của thuật toán có phần cao hơn với thuật toán có sẵn trong thư viện

In [ ]:

```

```