Python TASK 5 (All exercises in one)

Name- Jatin Karthik

Matriculation no.-313301

In [678]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as math
import random
from mpl_toolkits import mplot3d
```

EXERCISE-1 --Backward search for variable selection

In [719]:
```python
bank=pd.read_csv("bank.csv", delimiter=';')
bank.columns=["age","job","marital","education","default","balance","housing","loan","contact","day","month","duration","campaign","pdays","previous","poutcome","y"]
bank
```

Out[719]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	5	-1	0	unknown	no
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	1	-1	0	unknown	no
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	11	-1	0	unknown	no
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	4	211	3	other	no
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	2	249	7	other	no

4521 rows × 17 columns

In [720]:
```python
bank.corr()
```

Out[720]:

	age	balance	day	duration	campaign	pdays	previous
age	1.000000	0.083820	-0.017853	-0.002367	-0.005148	-0.008894	-0.003511
balance	0.083820	1.000000	-0.008677	-0.015950	-0.009976	0.009437	0.026196
day	-0.017853	-0.008677	1.000000	-0.024629	0.160706	-0.094352	-0.059114
duration	-0.002367	-0.015950	-0.024629	1.000000	-0.068382	0.010380	0.018080
campaign	-0.005148	-0.009976	0.160706	-0.068382	1.000000	-0.093137	-0.067833
pdays	-0.008894	0.009437	-0.094352	0.010380	-0.093137	1.000000	0.577562
previous	-0.003511	0.026196	-0.059114	0.018080	-0.067833	0.577562	1.000000

Convert any non-numeric values to numeric values. For example you can replace a country name with an integer value or more appropriately use hot-one encoding. [Hint: use hashmap (dict) or pandas.get_dummies]. Please explain your solution.

I did both using categorical values and using pd.dummies just to check on how the dataset performs

In [721]:
```python
columns=['job','marital','education','default','housing','loan','month','contact','duration','poutcome','y']
bank[columns] = bank[columns].astype('category')
bank.dtypes
```

Out[721]:
```
age             int64
job          category
marital      category
education    category
default      category
balance         int64
housing      category
loan         category
contact      category
day             int64
month        category
duration     category
campaign        int64
pdays           int64
previous        int64
poutcome     category
y            category
dtype: object
```

In [722]:
```python
bank_enc = bank.apply(lambda x: x.cat.codes if x.dtype.name == 'category' else x)
bank_enc.corr()
```

Out[722]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
age	1.000000	-0.021500	-0.381485	-0.121613	-0.017885	0.083820	-0.193888	-0.011250	0.015161	-0.017853	-0.040714	-0.006735	-0.005148	-0.008894	-0.003511	-0.009320	0.045092
job	-0.021500	1.000000	0.069390	0.170160	0.008324	0.009797	-0.128353	-0.040245	-0.074068	0.012865	-0.096613	-0.007960	-0.002739	-0.022760	0.005029	0.013049	0.027401
marital	-0.381485	0.069390	1.000000	0.102714	-0.020745	0.024971	-0.029851	-0.045210	-0.068236	0.008794	-0.035855	0.005323	0.005915	0.017050	0.038028	-0.027716	0.015042
education	-0.121613	0.170160	0.102714	1.000000	-0.010534	0.057725	-0.087070	-0.054086	-0.110554	0.014926	-0.050086	-0.013823	-0.001723	0.012077	0.023983	-0.032135	0.042987
default	-0.017885	0.008324	-0.020745	-0.010534	1.000000	-0.070886	0.006881	0.063994	0.008448	-0.013261	0.014297	-0.013535	-0.012348	-0.026317	-0.026656	0.039032	0.001303
balance	0.083820	0.009797	0.024971	0.057725	-0.070886	1.000000	-0.050227	-0.071349	-0.009665	-0.008677	0.023113	-0.009185	-0.009976	0.009437	0.026196	-0.029268	0.017905
housing	-0.193888	-0.128353	-0.029851	-0.087070	0.006881	-0.050227	1.000000	0.018451	0.196454	-0.031291	0.266630	0.017126	-0.003574	0.116893	0.038621	-0.093093	-0.104683
loan	-0.011250	-0.040245	-0.045210	-0.054086	0.063994	-0.071349	0.018451	1.000000	-0.007319	-0.004879	0.016329	-0.005424	0.017120	-0.031086	-0.022115	0.027028	-0.070517
contact	0.015161	-0.074068	-0.068236	-0.110554	0.008448	-0.009665	0.196454	-0.007319	1.000000	-0.033807	0.370077	-0.022171	0.012278	-0.243223	-0.187232	0.267173	-0.133595
day	-0.017853	0.012865	0.008794	0.014926	-0.013261	-0.008677	-0.031291	-0.004879	-0.033807	1.000000	-0.014795	-0.031781	0.160706	-0.094352	-0.059114	0.073714	-0.011244
month	-0.040714	-0.096613	-0.035855	-0.050086	0.014297	0.023113	0.266630	0.016329	0.370077	-0.014795	1.000000	0.005488	-0.108915	0.033292	0.046899	-0.030189	-0.040933
duration	-0.006735	-0.007960	0.005323	-0.013823	-0.013535	-0.009185	0.017126	-0.005424	-0.022171	-0.031781	0.005488	1.000000	-0.091937	0.014985	0.023232	-0.004120	0.406754
campaign	-0.005148	-0.002739	0.005915	-0.001723	-0.012348	-0.009976	-0.003574	0.017120	0.012278	0.160706	-0.108915	-0.091937	1.000000	-0.093137	-0.067833	0.110703	-0.061147
pdays	-0.008894	-0.022760	0.017050	0.012077	-0.026317	0.009437	0.116893	-0.031086	-0.243223	-0.094352	0.033292	0.014985	-0.093137	1.000000	0.577562	-0.859245	0.104087
previous	-0.003511	0.005029	0.038028	0.023983	-0.026656	0.026196	0.038621	-0.022115	-0.187232	-0.059114	0.046899	0.023232	-0.067833	0.577562	1.000000	-0.636372	0.116714
poutcome	-0.009320	0.013049	-0.027716	-0.032135	0.039032	-0.029268	-0.093093	0.027028	0.267173	0.073714	-0.030189	-0.004120	0.110703	-0.859245	-0.636372	1.000000	-0.082632
y	0.045092	0.027401	0.015042	0.042987	0.001303	0.017905	-0.104683	-0.070517	-0.133595	-0.011244	-0.040933	0.406754	-0.061147	0.104087	0.116714	-0.082632	1.000000

---OR---

In [723]:
```python
#bank2 = pd.get_dummies(bank[['job','marital','education','default','housing','loan','month','poutcome']])

#Function to encode the non numeric columns

def encode_and_bind(original_df, feature_to_encode):
    dummies = pd.get_dummies(original_df[[feature_to_encode]])
    original_df = pd.concat([original_df, dummies], axis=1)
    return(original_df)

A1=encode_and_bind(bank, 'job')
A2=encode_and_bind(A1, 'marital')
A3=encode_and_bind(A2, 'education')
A4=encode_and_bind(A3, 'default')
A5=encode_and_bind(A4, 'housing')
A6=encode_and_bind(A5, 'loan')
A7=encode_and_bind(A6, 'month')
A8=encode_and_bind(A7, 'poutcome')
A9=encode_and_bind(A8,'contact')


A9.loc[A9['y'] == 'yes', 'y_num'] = '1'
A9.loc[A9['y'] == 'no', 'y_num'] = '0'
A9=A9.drop(['y'],axis=1)
A9
```

Out[723]:

	age	job	marital	education	default	balance	housing	loan	contact	day	...	month_oct	month_sep	poutcome_failure	poutcome_other	poutcome_success	poutcome_unknown	contact_cellular	contact_telephone	contact_unknown	y_num
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	...	1	0	0	0	0	1	1	0	0	0
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	...	0	0	1	0	0	0	1	0	0	0
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	...	0	0	1	0	0	0	1	0	0	0
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	...	0	0	0	0	0	1	0	0	1	0
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	...	0	0	0	0	0	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	...	0	0	0	0	0	1	1	0	0	0
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	...	0	0	0	0	0	1	0	0	1	0
4518	57	technician	married	secondary	no	295	no	no	cellular	19	...	0	0	0	0	0	1	1	0	0	0
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	...	0	0	0	1	0	0	1	0	0	0
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	...	0	0	0	1	0	0	1	0	0	0

4521 rows × 61 columns

Finally, I stuck with the pd.dummies values because with more numbers, the algorithm might get biased with one frequently occuring number and the model might not perform well.

In [634]:
```python
A9.dropna()
```

Out[634]:

	age	job	marital	education	default	balance	housing	loan	contact	day	...	month_oct	month_sep	poutcome_failure	poutcome_other	poutcome_success	poutcome_unknown	contact_cellular	contact_telephone	contact_unknown	y_num
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	...	1	0	0	0	0	1	1	0	0	0
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	...	0	0	1	0	0	0	1	0	0	0
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	...	0	0	1	0	0	0	1	0	0	0
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	...	0	0	0	0	0	1	0	0	1	0
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	...	0	0	0	0	0	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	...	0	0	0	0	0	1	1	0	0	0
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	...	0	0	0	0	0	1	0	0	1	0
4518	57	technician	married	secondary	no	295	no	no	cellular	19	...	0	0	0	0	0	1	1	0	0	0
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	...	0	0	0	1	0	0	1	0	0	0
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	...	0	0	0	1	0	0	1	0	0	0

4521 rows × 61 columns

In [ ]:

In [724]:
```python
final=A9.drop(['job','marital','education','default','housing','loan','month','poutcome','contact','y_num'],axis=1)
final
```

Out[724]:

	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	...	month_nov	month_oct	month_sep	poutcome_failure	poutcome_other	poutcome_success	poutcome_unknown	contact_cellular	contact_telephone	contact_unknown
0	30	1787	19	79	1	-1	0	0	0	0	...	0	1	0	0	0	0	1	1	0	0
1	33	4789	11	220	1	339	4	0	0	0	...	0	0	0	1	0	0	0	1	0	0
2	35	1350	16	185	1	330	1	0	0	0	...	0	0	0	1	0	0	0	1	0	0
3	30	1476	3	199	4	-1	0	0	0	0	...	0	0	0	0	0	0	1	0	0	1
4	59	0	5	226	1	-1	0	0	1	0	...	0	0	0	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	-333	30	329	5	-1	0	0	0	0	...	0	0	0	0	0	0	1	1	0	0
4517	57	-3313	9	153	1	-1	0	0	0	0	...	0	0	0	0	0	0	1	0	0	1
4518	57	295	19	151	11	-1	0	0	0	0	...	0	0	0	0	0	0	1	1	0	0
4519	28	1137	6	129	4	211	3	0	1	0	...	0	0	0	0	1	0	0	1	0	0
4520	44	1136	3	345	2	249	7	0	0	1	...	0	0	0	0	1	0	0	1	0	0

4521 rows × 51 columns

In [725]:
```python
final=np.array(final)
```

While normalizing we change the scale of the data. Data is most commonly rescaled to fall between 0-1. In other words, for each sample from the dataset, we subtract the mean and divide by the standard deviation. By removing the mean from each sample, we effectively move the samples towards a mean of 0 (after all, we removed it from all samples). In addition, by dividing by the standard deviation, we yield a dataset where the values describe by how much of the standard deviation they are offset from the mean.

Normalize (Standardize) the data

In [726]:
```python
standardized_final = (final - final.mean()) / (final.std())
print(standardized_final)

standardized_final=pd.DataFrame(standardized_final)


#using categorical values
standardised_df=  (bank_enc-bank_enc.mean()) / (bank_enc.std())
print(standardised_df)
```

```
[[-1.11378718e-02  3.74357886e+00 -3.46449203e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]
 [-4.72685862e-03  1.01588661e+01 -5.17409555e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]
 [-4.52849817e-04  2.80970794e+00 -4.10559335e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]
 ...
 [ 4.65612470e-02  5.55168295e-01 -3.46449203e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]
 [-1.54118806e-02  2.35452600e+00 -6.24259775e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]
 [ 1.87801898e-02  2.35238900e+00 -6.88369907e-02 ... -7.31109995e-02
  -7.52480039e-02 -7.52480039e-02]]
           age       job   marital  education   default   balance   housing  \
0    -1.056153  1.716614 -0.246402  -1.644573 -0.130744  0.121058 -1.141925
1    -0.772497  0.795158 -0.246402  -0.309004 -0.130744  1.118521  0.875521
2    -0.583394 -0.126299  1.421239   1.026566 -0.130744 -0.024142  0.875521
3    -1.056153 -0.126299 -0.246402   1.026566 -0.130744  0.017724  0.875521
4     1.685850 -1.047755 -0.246402  -0.309004 -0.130744 -0.472701  0.875521
...        ...       ...       ...        ...       ...       ...       ...
4516 -0.772497  0.795158 -0.246402  -0.309004 -0.130744 -0.583345  0.875521
4517  1.496746  0.488006 -0.246402   1.026566  7.646823 -1.573497  0.875521
4518  1.496746  1.409462 -0.246402  -0.309004 -0.130744 -0.374682 -1.141925
4519 -1.245256 -1.047755 -0.246402  -0.309004 -0.130744 -0.094914 -1.141925
4520  0.267573 -0.740603  1.421239   1.026566 -0.130744 -0.095247  0.875521

          loan   contact       day     month  duration  campaign     pdays  \
0    -0.424709 -0.723561  0.374011  1.485250 -0.847033 -0.576766 -0.407173
1     2.354032 -0.723561 -0.595961  0.819197 -0.128964 -0.576766  2.988713
2    -0.424709 -0.723561  0.010271 -1.845016 -0.307208 -0.576766  2.898822
3     2.354032  1.494968 -1.565932  0.153144 -0.235910  0.387925 -0.407173
4    -0.424709  1.494968 -1.323439  0.819197 -0.098408 -0.576766 -0.407173
...        ...       ...       ...       ...       ...       ...       ...
4516 -0.424709 -0.723561  1.707721 -0.179883  0.426139  0.709488 -0.407173
4517  2.354032  1.494968 -0.838453  0.819197 -0.470174 -0.576766 -0.407173
4518 -0.424709 -0.723561  0.374011 -1.511990 -0.480359  2.638868 -0.407173
4519 -0.424709 -0.723561 -1.202193 -0.845936 -0.592398  0.387925  1.710262
4520  2.354032 -0.723561 -1.565932 -1.845016  0.507622 -0.255202  2.089802

      previous  poutcome         y
0    -0.320377  0.444364 -0.360862
1     2.041508 -2.579675 -0.360862
2     0.270094 -2.579675 -0.360862
3    -0.320377  0.444364 -0.360862
4    -0.320377  0.444364 -0.360862
...        ...       ...       ...
4516 -0.320377  0.444364 -0.360862
4517 -0.320377  0.444364 -0.360862
4518 -0.320377  0.444364 -0.360862
4519  1.451037 -1.571662 -0.360862
4520  3.812922 -1.571662 -0.360862

[4521 rows x 17 columns]
```

Split the data into a train/test splits according to the ratios 80%:20%

In [727]:
```python
training_data = standardized_final.sample(frac=0.8, random_state=3116)   #80% Train
testing_data = standardized_final.drop(training_data.index)          #20% test
#testing_data = standardized_final.sample(frac=0.2,random_state=3116)

print(f"No. of training examples:{training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")

X=np.array(training_data, dtype=float)
Xt=np.array(testing_data, dtype=float)
y=A9.iloc[:3617,60:]
Y= np.array(y, dtype=float)
yt=A9.iloc[3617:,60:]
Ytest=np.array(yt, dtype=float)
print(np.shape(X))
print(np.shape(Xt))


#for categorical values
training_data2=standardised_df.sample(frac=0.8, random_state=3116)
testing_data2 = standardised_df.drop(training_data.index)

Xs=np.array(training_data2.iloc[:3617,:16], dtype=float)
ys= bank_enc.iloc[:3617,16:]
Ys=np.array(ys, dtype=float)
```

```
No. of training examples:3617
No. of testing examples: 904
(3617, 51)
(904, 51)
```

Mini batch Gradient Acent- Parameters are updated after computing the gradient of error with respect to a subset of the training set

In [685]:
```python
def sigmoid(f):
    s= 1/(1+np.exp((-1)*f))
    return s

def hypothesis(X, theta):
    return np.dot(X, theta)

# function to compute gradient of error function w.r.t. theta
def gradient(X, Y, theta):
    #h = hypothesis(X, theta)
    y_cap=sigmoid(X.dot(theta))
    grad = np.dot(X.transpose(), (y_cap - Y))
    grad=grad*(-1)
    #print(grad)
    return grad

# Gradient ascent using mini batching
def create_batches(X, Y, batch_size):
    mini_batches = []
    data = np.hstack((X, Y))
    np.random.shuffle(data)
    n_minibatches = data.shape[0] // batch_size   #integer division for no of columns to remain an integer
    i = 0

    for i in range(n_minibatches + 1):
        mini_batch = data[i * batch_size:(i + 1)*batch_size, :]
        X_mini = mini_batch[:, :-1]
        Y_mini = mini_batch[:, -1].reshape((-1, 1))
        mini_batches.append((X_mini, Y_mini))
        #condition if the number of values of features is not divisible by the batch size completely
    if data.shape[0] % batch_size != 0:
        mini_batch = data[i * batch_size:data.shape[0]]
        X_mini = mini_batch[:, :-1]
        Y_mini = mini_batch[:, -1].reshape((-1, 1))
        mini_batches.append((X_mini, Y_mini))
    return mini_batches

#Function for GA using logistic regression
def GA_Logisticreg(X, Y, learning_rate = 0.001, batch_size = 16):
    theta = np.zeros((X.shape[1], 1))

    max_iters = 5
    accuracys=[]
    L=np.sum((Y.T).dot(X.dot(theta)) - np.log(1+np.exp(X.dot(theta))))
    for itr in range(max_iters):
        mini_batches = create_batches(X, Y, batch_size)

        for mini_batch in mini_batches:
            X_mini, Y_mini = mini_batch

            theta = theta + learning_rate * gradient(X_mini, Y_mini, theta)
            #error_list.append(cost(X_mini, y_mini, theta))

    pred_xtrain=sigmoid(X.dot(theta))
    pred_xtrain=np.array(pred_xtrain,dtype=float)

    for c,row in enumerate(pred_xtrain):
        if row<0.5:
            pred_xtrain[c]=0
        else:
            pred_xtrain[c]=1

    a=classification_accu(Y,pred_xtrain)
    accuracys.append(a)
    return theta,L,accuracys #L # error_list
```

In [686]:
```python
def classification_accu(true, pred):
    N = true.shape[0]
    accuracy = ((true == pred).sum()) / N
    return accuracy*100
```

In [687]:
```python
t,l,A=GA_Logisticreg(X,Y)
```

In [688]:
```python
#scoring metric for backward search
def AIC(L,p):
    #L=np.sum((Y.T).dot(X.dot(B)) - np.log(1+np.exp(X.dot(B))))
    Aic=(-2)*(np.log(L))+2*p
    return Aic
```

In [689]:
```python
#implementing the pseudo code from Prof. Lars slides below

def Back_search(X,Y,Xt,Ytest):
    V=list(range(51))
    theta=np.zeros([51,1],dtype=float)
    b1,l1,a1=GA_Logisticreg(X,Y)
    b2,l2,a2=GA_Logisticreg(Xt,Ytest)
    e_allbest=AIC(l1,len(Y))-AIC(l2,len(Ytest))
    vbest=1

    while vbest!=0:
        vbest=0
        ebest=e_allbest
        for v in V:
            V_dash=V.remove(v)
            y_cap=sigmoid(X.dot(b1))
            L=np.sum((y_cap.T).dot(X.dot(theta)) - np.log(1+np.exp(X.dot(theta))))
            e=AIC(L,len(y_cap))-AIC(l2,len(Xt))
            if e<ebest:
                vbest=v
                ebest=e
        if ebest<e_allbest:
            V=V.remove(vbest)
            e_allbest=ebest
    print(V)
    return V
```

In [690]:
```python
a=Back_search(X,Y,Xt,Ytest)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]
<ipython-input-688-7610e32a4baa>:4: RuntimeWarning: invalid value encountered in log
  Aic=(-2)*(np.log(L))+2*p
```

In [700]:
```python
#final=pd.DataFrame(final)
colname = final.columns[a]
print ("The list of the features remaining after Backward search feature selection is",colname)
```

```
The list of the features remaining after Backward search feature selection is Index(['balance', 'duration', 'pdays', 'job_admin.', 'job_entrepreneur',
       'job_management', 'job_self-employed', 'job_student', 'job_unemployed',
       'marital_divorced', 'marital_single', 'education_secondary',
       'education_unknown', 'default_yes', 'housing_yes', 'loan_yes',
       'month_aug', 'month_feb', 'month_jul', 'month_mar', 'month_nov',
       'month_sep', 'poutcome_other', 'poutcome_unknown', 'contact_telephone'],
      dtype='object')
```

In [625]:
```python
b,ac_final,l,acc=GA_Logisticreg2(Xt[:,a],Ytest,0.0001,0.001,batch_size=50)
print("Final accuracy on test set",ac_final,"% and error is ",100-ac_final)
```

```
Final accuracy on test set 83.4070796460177 % and error is  16.592920353982294
```

Exercise 2--Regularization for Logistic Regression

The general procedure of K cross validation grid search is as follows:

Shuffle the dataset randomly.

Split the dataset into k groups

For each unique group: (1)Take the group as a hold out or test data set (2)Take the remaining groups as a training data set (3)Fit a model on the training set and evaluate it on the test set (4)Retain the evaluation score and discard the model

Summarize the skill of the model using the sample of model evaluation scores

Pick a range of α0 and λ defined on grid. You can choose fixed batchsize = 50.

In [556]:
```python
def cross_validation(dataset,k): #different datasets i.e Xtrain, Ytrain will be passed and various combinations
                                 #of test and train would be formed called folds
        data=[]
        #data_copy=list(dataset)
        dataset = dataset.sample(frac = 1)            # shuffle the DataFrame rows

        k_size= int(len(dataset)/k)
        counter=0

        for i in range(0,k):
            k_list=pd.DataFrame(dataset.head(0))
            for j in range(i*k_size,(i*k_size)+k_size):
                k_list=k_list.append(dataset.iloc[j])
                counter=counter+1
            data.append(k_list)

        #for the remaining rows of data which were not divisble completely
        for m in range(counter,len(dataset)):
            data[k-1]=data[k-1].append(dataset.iloc[m])

        return data

def cross_valid_holdout(X_cv,Y_cv,k):
        final_list=[]

        for n in range(0,k):

            xtest=X_cv[n]
            ytest=Y_cv[n]
            x_train=pd.DataFrame()
            y_train=pd.DataFrame()
            for p in range(0,k):
                if n!=p:
                    x_train=x_train.append(X_cv[p])
                    y_train=y_train.append(Y_cv[p])
            final=dict([('X',x_train),('Y',y_train),('Xt',xtest),('Yt',ytest)])
            final_list.append(final)

        return final_list
```

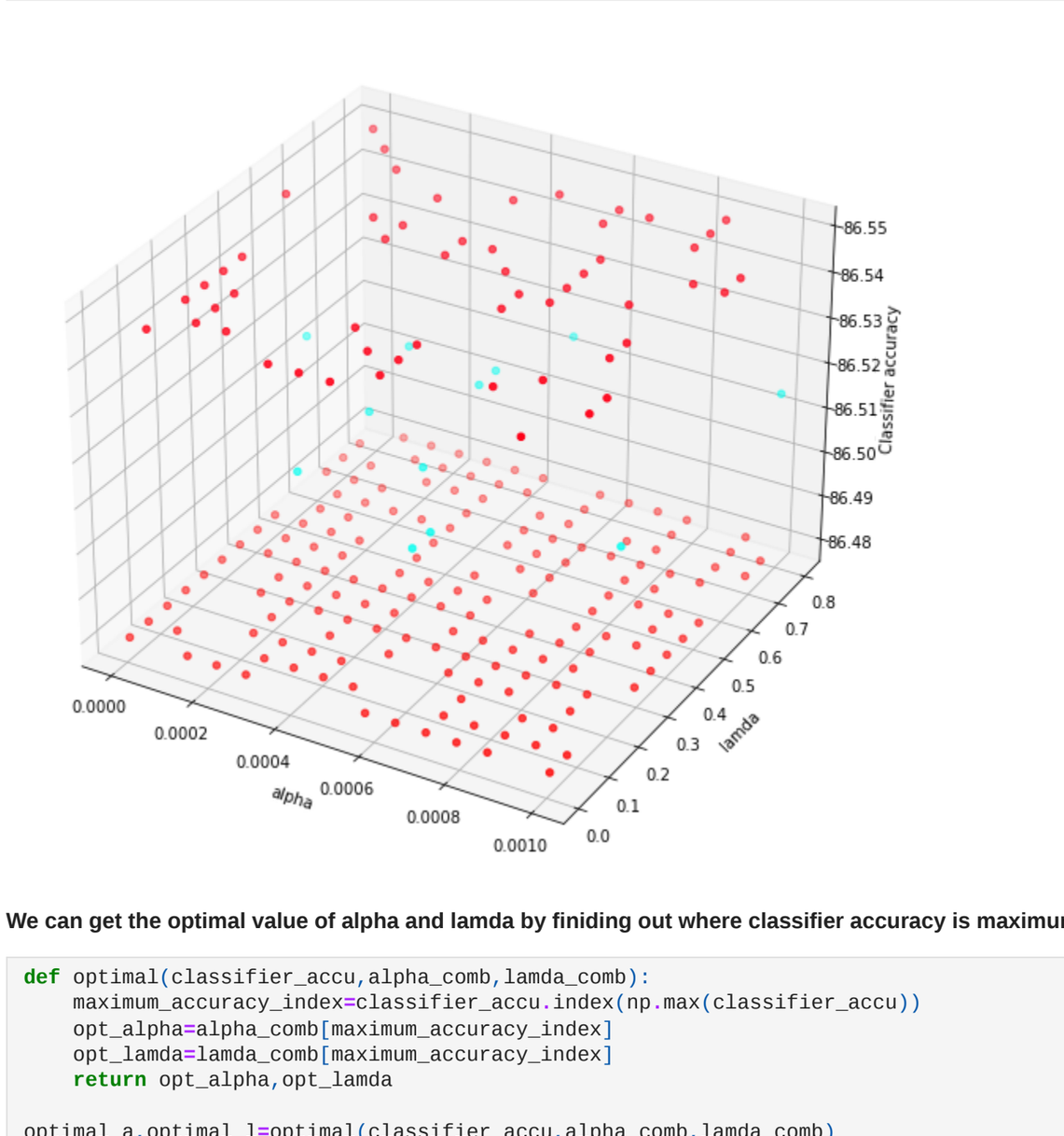
Implement k-fold cross-validation protocol for grid search. For each combination of α0 and λ you will perform k-fold cross-validation. Let k = 5 in this case.

In [702]:
```python
x_kcross=cross_validation(training_data,k=5)
y_kcross=cross_validation(y,k=5)
#data_final=cross_valid_holdout(x_kcross,y_kcross,k=5)


#try changing these

alpha=np.linspace(0.000000001,0.001,15)
lamda=np.linspace(0.01,0.8,15)
class_accu=[]
epochs=20
k=5

def gridsearch(alpha,lamda):
    combination_list=[]
    for i in range(0,len(alpha)):
        for k in range(0,len(lamda)):
            combination_list.append(dict([('alpha',alpha[i]),('lamda',lamda[k])]))
    return combination_list

gs=gridsearch(alpha,lamda)
classifier_accu=[]
alpha_comb=[]
lamda_comb=[]


for i in range (0,len(gs)):
    data_final=cross_valid_holdout(x_kcross,y_kcross,k=5)
    #data_final=np.array(data_final)

    for j in range(0,k):
        t_1,L1,a_1=GA_Logisticreg(np.array(data_final[j]['X'],dtype=float),np.array(data_final[j]['Y'],dtype=float),0.0001,batch_size = 50)


    alpha_comb.append(gs[i]['alpha'])
    lamda_comb.append(gs[i]['lamda'])
    classifier_accu.append(a_1)
```

1. Keep track of mean performance (i.e. Classification Accuracy value) across k − folds for each set of hyperparameters. Plot on the grid α0 vs λ the Classification Accuracy score for all combinations. [Hint: you can use a 3D plot with axes=α0, λ, Acc.]

In the above alpha and lamda range, I tried various ranges but in my opinion values bigger for lamda as compared to alpha worked for the better. They gave better classification accuracies.

In [703]:
```python
fig = plt.figure()
fig.set_figheight(10)
fig.set_figwidth(10)
ax = plt.axes(projection= '3d')

ax.scatter3D(alpha_comb, lamda_comb, classifier_accu, c=classifier_accu, cmap='hsv')
ax.set_xlabel('alpha')
ax.set_ylabel('lamda')
ax.set_zlabel('Classifier accuracy')
plt.show()
```



We can get the optimal value of alpha and lamda by finiding out where classifier accuracy is maximum.

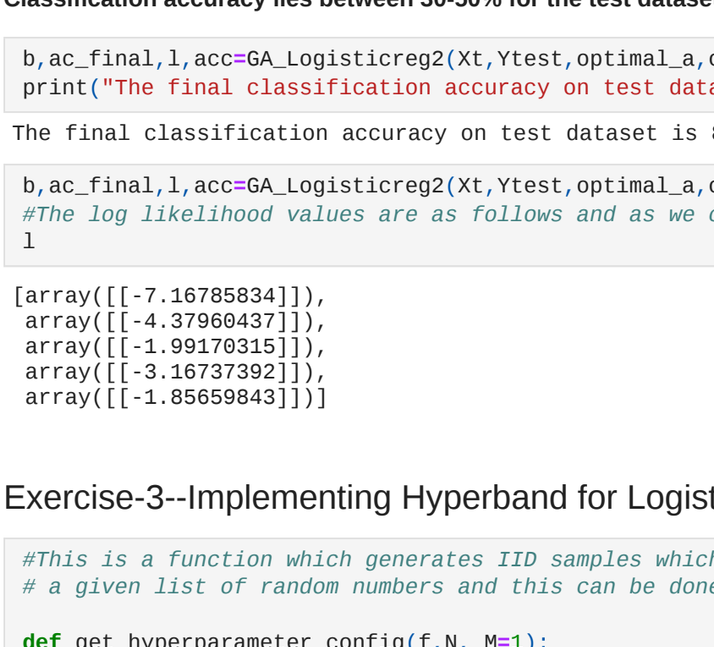
In [704]:
```python
def optimal(classifier_accu,alpha_comb,lamda_comb):
    maximum_accuracy_index=classifier_accu.index(np.max(classifier_accu))
    opt_alpha=alpha_comb[maximum_accuracy_index]
    opt_lamda=lamda_comb[maximum_accuracy_index]
    return opt_alpha,opt_lamda

optimal_a,optimal_l=optimal(classifier_accu,alpha_comb,lamda_comb)
print("The optimal aplha and lambda values are:",optimal_a,"and",optimal_l)
```

```
The optimal aplha and lambda values are: 1e-09 and 0.5178571428571429
```

In [705]:
```python
betas = lambda x,y,beta,alpha,lamda : beta-alpha*(-2*np.dot(x.T,y-sigmoid(x.dot(beta)))+(2*lamda)*beta)

def GA_Logisticreg2(X, Y, alpha, lamda, batch_size=50):
    beta = np.zeros((X.shape[1], 1))

    #error_list = []
    max_iters = 5
    accuracys=[]
    log_likelihood=[]

    for itr in range(max_iters):
        mini_batches = create_batches(X, Y, batch_size)

        for mini_batch in mini_batches:
            X_mini, Y_mini = mini_batch

            beta = beta + alpha * gradient(X_mini, Y_mini, beta)


        beta_hat = betas(X_mini,Y_mini,beta,alpha,lamda)
        pred_iterations=np.array(sigmoid(X_mini.dot(beta_hat)))
        for p,row in enumerate(pred_iterations):
            if row<0.5:
                pred_iterations[p]=0
            else:
                pred_iterations[p]=1

        accuracy_iteration=classification_accu(Y_mini,pred_iterations)
        accuracys.append(accuracy_iteration)
        regularization=lamda*((beta_hat.T).dot(beta_hat))
        LL=np.sum((Y_mini.T).dot(X_mini.dot(beta)) - np.log(1+np.exp(X_mini.dot(beta))))-regularization
        log_likelihood.append(LL)
    pred_xtrain=sigmoid(X.dot(beta_hat))
    pred_xtrain=np.array(pred_xtrain,dtype=float)

    for c,row in enumerate(pred_xtrain):
        if row<0.5:
            pred_xtrain[c]=0
        else:
            pred_xtrain[c]=1

    accu_final=classification_accu(Y,pred_xtrain)

#     regularization=lamda*((beta_hat.T).dot(beta_hat))
#     LL=np.sum((Y.T).dot(X.dot(beta)) - np.log(1+np.exp(X.dot(beta))))-regularization
    return beta_hat,accu_final,log_likelihood,accuracys
```

In [392]:
```python
b,ac_final,ll,acc_train=GA_Logisticreg2(Xt,Ytest,optimal_a,optimal_l,batch_size=50)
plt.xlabel('k-folds')
plt.ylabel('classification accuracy')
plt.plot(list(range(5)),acc_train)
#graph for accuracy per k-fold
```

Out[392]: [<matplotlib.lines.Line2D at 0x273f38f89a0>]



Classification accuracy lies between 30-50% for the test dataset.

In [706]:
```python
b,ac_final,l,acc=GA_Logisticreg2(Xt,Ytest,optimal_a,optimal_l,batch_size=50)
print("The final classification accuracy on test dataset is",ac_final)
```

```
The final classification accuracy on test dataset is 82.52212389380531
```

In [415]:
```python
b,ac_final,l,acc=GA_Logisticreg2(Xt,Ytest,optimal_a,optimal_l,batch_size=50)
#The log likelihood values are as follows and as we can see there is a decreasing trend of values
l
```

Out[415]:
```
[array([[-7.16785834]]),
 array([[-4.37960437]]),
 array([[-1.99170315]]),
 array([[-3.16737392]]),
 array([[-1.85659843]])]
```

Exercise-3--Implementing Hyperband for Logistic Regression

In [731]:
```python
#This is a function which generates IID samples which are independent identical random distribution variables which produces
# a given list of random numbers and this can be done with np.random too but we need a pdf function defining these IID samples

def get_hyperparameter_config(f,N, M=1):
    sample = M*np.random.random(N)
    y=[]
    sum = 0
    for x in sample:
        v = f(x);
        sum+=v;
        y.append(v)
    y = y/sum;
    return np.random.choice(sample, p=y, size=N)

def fn(x):
    return x**1.25

#n=get_hyperparameter_config(fn,100)

training_data = standardized_final.sample(frac=0.7, random_state=3116)   #70% Train
#validation_data=standardized_final.drop(training_data.index)
validation_data = standardized_final.sample(frac=0.15,random_state=3116) #15% validation set
#testing_data=validation_data.drop(validation_data.index)
testing_data = standardized_final.sample(frac=0.15,random_state=3116)   #15% test set

print(np.shape(validation_data))

Xv=np.array(validation_data,dtype=float)
Yv=A9.iloc[:678,60:]
Yv=np.array(Yv,dtype=float)
print(np.shape(Xv))
```

```
(678, 51)
(678, 51)
```

In [732]:
```python
#this function will return the loss for the specific columns as defined from the main hyperband function ri

def run_thenreturn_val_loss(ri,T):
    a,b,Loss,d=GA_Logisticreg2(Xv[:,ri:ri], Yv, T[ri],T[ri], batch_size=50)
    return Loss
```

In [733]:
```python
#this function looks into the losses and returns the features with least loss nd does this until only one feature remains
def Top_k(L,reducer):
    #successive halving
    #print(reducer)
    L.sort()
    reducer=int(reducer)
    index_stop=int(len(L)/reducer)
    L=L[:index_stop]
    return L
```

In [734]:
```python
import math

def hyperband_algo(R,eta=3): #eta=3 by default as given in the algorithm
    # all the notations are taken from the hyperband algorithm given in the paper

    s_max=np.log(R)
    B=(s_max+1)*R    #Batch size
    s_max=math.ceil(s_max)
    for s in range(s_max,0,-1):
        n= (B/R)*((eta**s)/(s+1))
        r=R*(eta**(-s))
        T=get_hyperparameter_config(fn,100)
        #print(s)
        for i in range(s):
            ni=math.floor(n*(n**(-i)))
            ri=int(r*(n**(-i)))
            print("the best found hyperparameters are",ri) #best hyperparameter columns passed
            L=run_thenreturn_val_loss(ri,T)
            T=Top_k(L,eta)
    return T
```

In [735]:
```python
sl=hyperband_algo(R=81,eta=3)
```

```
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 1
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 3
the best found hyperparameters are 0
the best found hyperparameters are 0
the best found hyperparameters are 9
the best found hyperparameters are 0
the best found hyperparameters are 27
```

In [736]:
```python
#Taking the above 4 features and using only these hyperparameters for calculating accuracy
X_hyper=np.array(testing_data,dtype=float)
ytst=A9.iloc[:678,60:]
Y_hyper=np.array(ytst,dtype=float)

X_hyper1=X_hyper[:,0:1]
X_hyper1=np.hstack((X_hyper1,X_hyper[:,8:8]))
X_hyper1=np.hstack((X_hyper1,X_hyper[:,26:26]))

print(np.shape(Y_hyper))
a,b,c,d=GA_Logisticreg2(X_hyper1, Y_hyper, optimal_a,optimal_l, batch_size=50)
a
pred_xhyper=sigmoid(X_hyper1.dot(a))

pred_xhyper=np.array(pred_xhyper,dtype=float)

for c,row in enumerate(pred_xhyper):
    if row<0.5:
        pred_xhyper[c]=0
    else:
        pred_xhyper[c]=1

accu_final=classification_accu(Y_hyper,pred_xhyper)
print("Using the hyperband algorithm, the classification accuracy on test set is",accu_final)
```

```
(678, 1)
Using the hyperband algorithm, the classification accuracy on test set is 60.02949852507374
```

My final opinion is that since the dataset target is unbalanced with a ratio of 20% yes and 80% no. I think this has made a huge difference in effecting the classification accuracy. I have not stratified the dataset and maybe that is one of the main reasons that I am getting low classifcation accuracies.

References:- https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8 https://pbpython.com/categorical-encoding.html https://www.geeksforgeeks.org/ml-mini-batch-gradient-descent-with-python/

T-2 https://stats.stackexchange.com/questions/266225/step-by-step-explanation-of-k-fold-cross-validation-with-grid-search-to-optimise

T-3 https://stackoverflow.com/questions/59246090/how-to-generate-iid-sample-from-a-given-arbitrary-probability-density-function

In [ ]: