BUSINESS DATA SCIENCE

ASSIGNMENT-4

Problem 1:

weights $w = [1, -30, 3]$

$x_1 = [20, 0, 0]$, $y_1 = 1$

$x_2 = [23, 1, 1]$, $y_2 = 0$

$Logit = w^T x$

Customer 1 = Logit 1 $= [1, -30, 3] = w$

$$w^T = \begin{bmatrix} 1 \\ -30 \\ 3 \end{bmatrix}$$

$w^T x_1$

$$\begin{bmatrix} 1 \\ -30 \\ 3 \end{bmatrix} \begin{bmatrix} 20, 0, 0 \end{bmatrix}$$

$= 20 - 0 - 0$

$= 20$

$Logit 2 = w^T x_2$

$$\begin{bmatrix} 1 \\ -30 \\ 3 \end{bmatrix} \begin{bmatrix} 23, 1, 1 \end{bmatrix}$$

$= -4$

Probability of customer 1 clicking $= \frac{1}{1+e^{-20}}$

$= 0.99$

Probability of customers 2 clicking $= \frac{1}{1+e^{4}}$

$= 0.017$

Cross entropy loss

$H(p,q) = -\Sigma p \log q$

$p =$ [Probability that Customer 1 clicks $= 0.99$

Probability that Customer 1 does not click $= 0.01$]

$q =$ [probability that customer 2 clicks $= 0.017$

probability that customer 2 does not click $= 0.983$]

$$H(P,Q) = -\left[0.99 \log_2(0.017) + 0.01 \log_2(0.983)\right]$$

$$= -\left[0.99(-5.878) + 0.01(-0.024)\right]$$

$$= -\left[-5.81 - 0.00024\right]$$

$$= -\left[-5.81024\right]$$

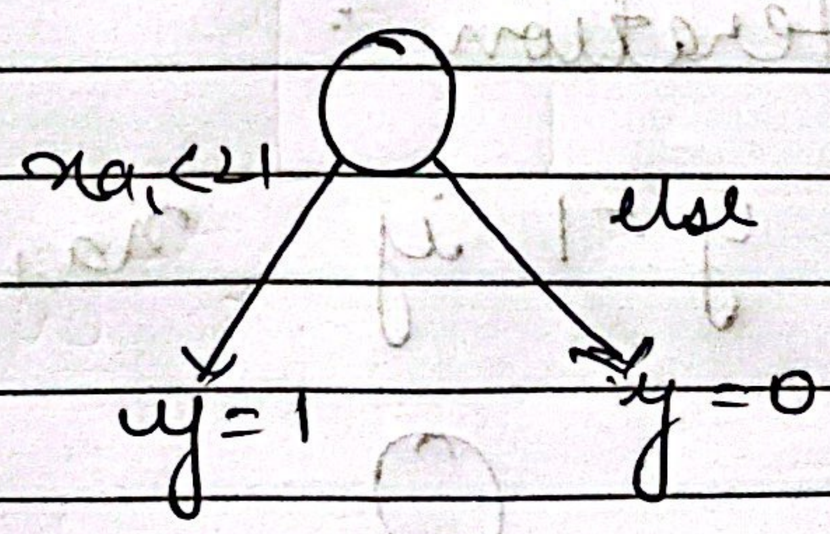
$$= 5.81024$$

Decision Tree of depth 1 :

Level 1 iteration :

$y = 1$ if $x_{a,1} < 21$

($x_{a,1}$ is the first feature of customer ID matrix)



This satisfies the dataset:

For customer 1, $x_{1,1} = 20$ and he clicked on the ad.

For customer 2, $x_{2,1} = 23$ and he did not click on the ad.

Gini Impurity

$$= 1 - \Sigma p^2$$

		p
Ad clicked	1	0.5
Ad not clicked	1	0.5

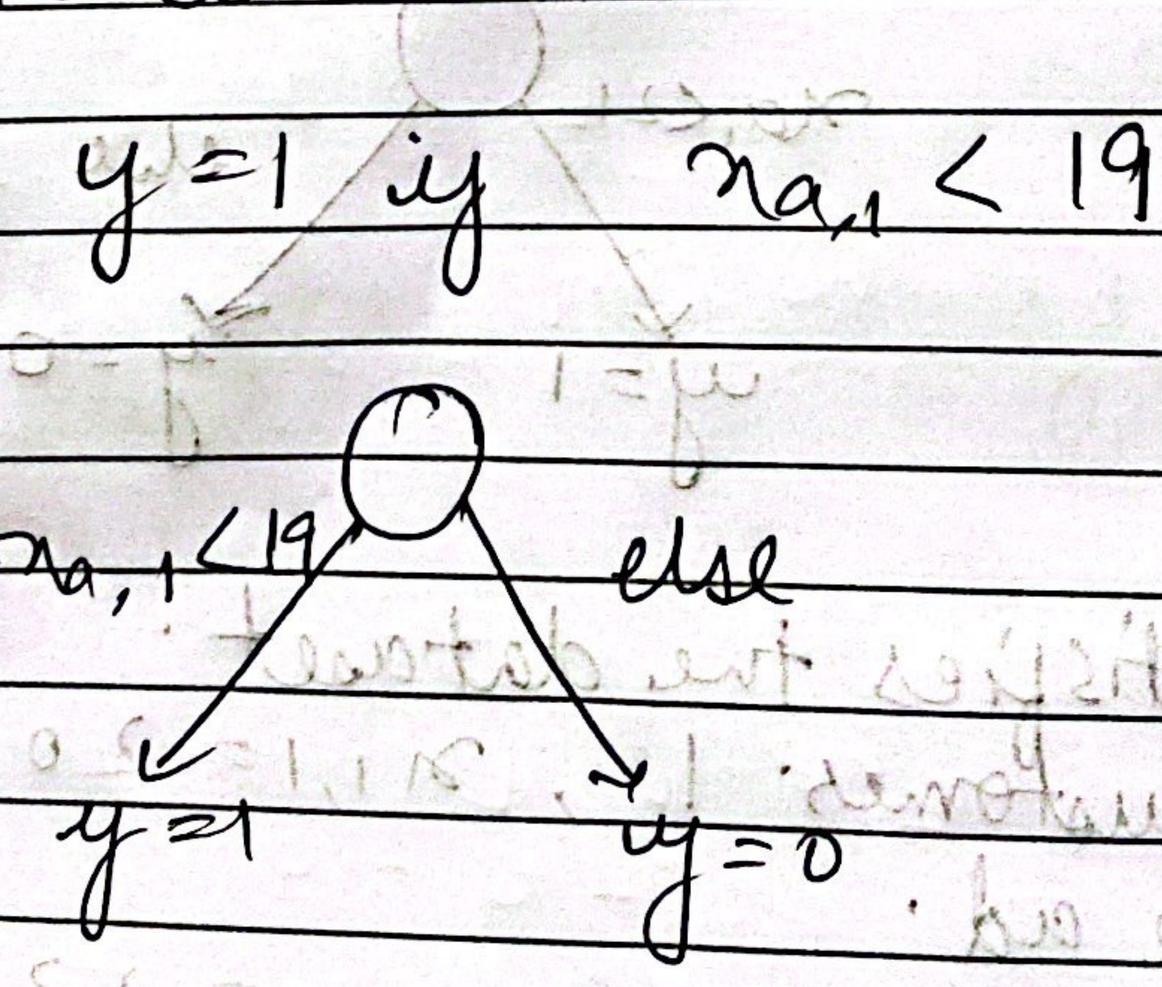
$$= 1 - \{(0.5)^2 + (0.5)^2\}$$

$$= 1 - 0.5$$

$= 0.5$ = Gini Impurity is very high

Now, designing decision tree on the first feature.

First Iteration :-



For customer 1, $x_{1,1} = 20$, $\therefore y = 0$

For customer 2, $x_{2,1} = 23$, $\therefore y = 0$

Gini Impurity

$= 1 - \Sigma p^2$

		p
Ad clicked	0	0
Ad not clicked	2	1

$= 1 - \{(0)^2 + (1)^2\}$

$= 1 - 1$

$= 0$ = Gini impurity is the lowest

It will not perform well if the no. of samples increased as it would result in overfitting.

In [1]:

```
### BDS Homework 4 - Problem 1 - CIFAR dataset
### &
### Problem 4 - CIFAR Continued
```

In [2]:

```
#Problem 1
import numpy as np
from sklearn.datasets import fetch_openml

#Use the fetch openml command from sklearn.datasets to import the CIFAR-10-Small data set

df = fetch_openml('CIFAR_10_small')

df
```

Out[2]:

```
{'data':            a0     a1     a2     a3     a4     a5     a6     a7     a8     a9  \
 0        59.0   43.0   50.0   68.0   98.0  119.0  139.0  145.0  149.0  149.0
 1       154.0  126.0  105.0  102.0  125.0  155.0  172.0  180.0  142.0  111.0
 2       255.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0
 3        28.0   37.0   38.0   42.0   44.0   40.0   40.0   24.0   32.0   43.0
 4       170.0  168.0  177.0  183.0  181.0  177.0  181.0  184.0  189.0  189.0
 ...       ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
 19995    76.0   76.0   77.0   76.0   75.0   76.0   76.0   76.0   76.0   78.0
 19996    81.0   91.0   98.0  106.0  108.0  110.0   80.0   84.0   88.0   90.0
 19997    20.0   19.0   15.0   15.0   14.0   13.0   12.0   11.0   10.0    9.0
 19998    25.0   15.0   23.0   17.0   23.0   51.0   74.0   91.0  114.0  137.0
 19999    73.0   98.0   99.0   77.0   59.0  146.0  214.0  176.0  125.0  218.0

         ...  a3062  a3063  a3064  a3065  a3066  a3067  a3068  a3069  a3070  \
 0       ...   59.0   58.0   65.0   59.0   46.0   57.0  104.0  140.0   84.0
 1       ...   22.0   42.0   67.0  101.0  122.0  133.0  136.0  139.0  142.0
 2       ...   78.0   83.0   80.0   69.0   66.0   72.0   79.0   83.0   83.0
 3       ...   53.0   39.0   59.0   42.0   44.0   48.0   38.0   28.0   37.0
 4       ...   92.0   88.0   85.0   82.0   83.0   79.0   78.0   82.0   78.0
 ...     ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
 19995   ...  228.0  185.0  177.0  223.0  239.0  239.0  235.0  236.0  234.0
 19996   ...  126.0  107.0  143.0  155.0  156.0  160.0  173.0  129.0  147.0
 19997   ...  114.0  112.0   68.0   50.0   52.0   52.0   51.0   50.0   53.0
 19998   ...   87.0   84.0   83.0   84.0   79.0   78.0   78.0   80.0   81.0
 19999   ...   84.0   89.0   88.0   85.0   93.0   93.0   90.0   94.0   58.0

        a3071
 0       72.0
 1      144.0
 2       84.0
 3       46.0
 4       80.0
 ...      ...
 19995  236.0
 19996  160.0
 19997   47.0
 19998   80.0
 19999   26.0

 [20000 rows x 3072 columns],
 'target': 0        6
 1         9
 2         9
 3         4
 4         1
          ..
 19995     8
 19996     3
 19997     5
```

```
19998    1
19999    7
Name: class, Length: 20000, dtype: category
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9'],
'frame':           a0     a1     a2     a3     a4     a5     a6     a7     a8     a9  \
0       59.0   43.0   50.0   68.0   98.0  119.0  139.0  145.0  149.0  149.0
1      154.0  126.0  105.0  102.0  125.0  155.0  172.0  180.0  142.0  111.0
2      255.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0  253.0
3       28.0   37.0   38.0   42.0   44.0   40.0   40.0   24.0   32.0   43.0
4      170.0  168.0  177.0  183.0  181.0  177.0  181.0  184.0  189.0  189.0
...      ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
19995   76.0   76.0   77.0   76.0   75.0   76.0   76.0   76.0   76.0   78.0
19996   81.0   91.0   98.0  106.0  108.0  110.0   80.0   84.0   88.0   90.0
19997   20.0   19.0   15.0   15.0   14.0   13.0   12.0   11.0   10.0    9.0
19998   25.0   15.0   23.0   17.0   23.0   51.0   74.0   91.0  114.0  137.0
19999   73.0   98.0   99.0   77.0   59.0  146.0  214.0  176.0  125.0  218.0

       ...  a3063  a3064  a3065  a3066  a3067  a3068  a3069  a3070  a3071  \
0      ...   58.0   65.0   59.0   46.0   57.0  104.0  140.0   84.0   72.0
1      ...   42.0   67.0  101.0  122.0  133.0  136.0  139.0  142.0  144.0
2      ...   83.0   80.0   69.0   66.0   72.0   79.0   83.0   83.0   84.0
3      ...   39.0   59.0   42.0   44.0   48.0   38.0   28.0   37.0   46.0
4      ...   88.0   85.0   82.0   83.0   79.0   78.0   82.0   78.0   80.0
...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
19995  ...  185.0  177.0  223.0  239.0  239.0  235.0  236.0  234.0  236.0
19996  ...  107.0  143.0  155.0  156.0  160.0  173.0  129.0  147.0  160.0
19997  ...  112.0   68.0   50.0   52.0   52.0   51.0   50.0   53.0   47.0
19998  ...   84.0   83.0   84.0   79.0   78.0   78.0   80.0   81.0   80.0
19999  ...   89.0   88.0   85.0   93.0   93.0   90.0   94.0   58.0   26.0

       class
0          6
1          9
2          9
3          4
4          1
...      ...
19995      8
19996      3
19997      5
19998      1
19999      7

[20000 rows x 3073 columns],
'categories': None,
'feature_names': ['a0',
 'a1',
 'a2',
 'a3',
 'a4',
 'a5',
 'a6',
 'a7',
 'a8',
 'a9',
 'a10',
 'a11',
 'a12',
 'a13',
 'a14',
 'a15',
 'a16',
 'a17',
 'a18',
 'a19',
 'a20',
 'a21',
 'a22',
 'a23',
 'a24',
 'a25',
 'a26',
 'a27',
```

'a27',
'a28',
'a29',
'a30',
'a31',
'a32',
'a33',
'a34',
'a35',
'a36',
'a37',
'a38',
'a39',
'a40',
'a41',
'a42',
'a43',
'a44',
'a45',
'a46',
'a47',
'a48',
'a49',
'a50',
'a51',
'a52',
'a53',
'a54',
'a55',
'a56',
'a57',
'a58',
'a59',
'a60',
'a61',
'a62',
'a63',
'a64',
'a65',
'a66',
'a67',
'a68',
'a69',
'a70',
'a71',
'a72',
'a73',
'a74',
'a75',
'a76',
'a77',
'a78',
'a79',
'a80',
'a81',
'a82',
'a83',
'a84',
'a85',
'a86',
'a87',
'a88',
'a89',
'a90',
'a91',
'a92',
'a93',
'a94',
'a95',
'a96',
'a97',
'a98',
'a99',

'a99',
'a100',
'a101',
'a102',
'a103',
'a104',
'a105',
'a106',
'a107',
'a108',
'a109',
'a110',
'a111',
'a112',
'a113',
'a114',
'a115',
'a116',
'a117',
'a118',
'a119',
'a120',
'a121',
'a122',
'a123',
'a124',
'a125',
'a126',
'a127',
'a128',
'a129',
'a130',
'a131',
'a132',
'a133',
'a134',
'a135',
'a136',
'a137',
'a138',
'a139',
'a140',
'a141',
'a142',
'a143',
'a144',
'a145',
'a146',
'a147',
'a148',
'a149',
'a150',
'a151',
'a152',
'a153',
'a154',
'a155',
'a156',
'a157',
'a158',
'a159',
'a160',
'a161',
'a162',
'a163',
'a164',
'a165',
'a166',
'a167',
'a168',
'a169',
'a170',
'a171'

'a171',
'a172',
'a173',
'a174',
'a175',
'a176',
'a177',
'a178',
'a179',
'a180',
'a181',
'a182',
'a183',
'a184',
'a185',
'a186',
'a187',
'a188',
'a189',
'a190',
'a191',
'a192',
'a193',
'a194',
'a195',
'a196',
'a197',
'a198',
'a199',
'a200',
'a201',
'a202',
'a203',
'a204',
'a205',
'a206',
'a207',
'a208',
'a209',
'a210',
'a211',
'a212',
'a213',
'a214',
'a215',
'a216',
'a217',
'a218',
'a219',
'a220',
'a221',
'a222',
'a223',
'a224',
'a225',
'a226',
'a227',
'a228',
'a229',
'a230',
'a231',
'a232',
'a233',
'a234',
'a235',
'a236',
'a237',
'a238',
'a239',
'a240',
'a241',
'a242',

'a243',
'a244',
'a245',
'a246',
'a247',
'a248',
'a249',
'a250',
'a251',
'a252',
'a253',
'a254',
'a255',
'a256',
'a257',
'a258',
'a259',
'a260',
'a261',
'a262',
'a263',
'a264',
'a265',
'a266',
'a267',
'a268',
'a269',
'a270',
'a271',
'a272',
'a273',
'a274',
'a275',
'a276',
'a277',
'a278',
'a279',
'a280',
'a281',
'a282',
'a283',
'a284',
'a285',
'a286',
'a287',
'a288',
'a289',
'a290',
'a291',
'a292',
'a293',
'a294',
'a295',
'a296',
'a297',
'a298',
'a299',
'a300',
'a301',
'a302',
'a303',
'a304',
'a305',
'a306',
'a307',
'a308',
'a309',
'a310',
'a311',
'a312',
'a313',
'a314',

'a315',
'a316',
'a317',
'a318',
'a319',
'a320',
'a321',
'a322',
'a323',
'a324',
'a325',
'a326',
'a327',
'a328',
'a329',
'a330',
'a331',
'a332',
'a333',
'a334',
'a335',
'a336',
'a337',
'a338',
'a339',
'a340',
'a341',
'a342',
'a343',
'a344',
'a345',
'a346',
'a347',
'a348',
'a349',
'a350',
'a351',
'a352',
'a353',
'a354',
'a355',
'a356',
'a357',
'a358',
'a359',
'a360',
'a361',
'a362',
'a363',
'a364',
'a365',
'a366',
'a367',
'a368',
'a369',
'a370',
'a371',
'a372',
'a373',
'a374',
'a375',
'a376',
'a377',
'a378',
'a379',
'a380',
'a381',
'a382',
'a383',
'a384',
'a385',
'a386',

'a387',
'a388',
'a389',
'a390',
'a391',
'a392',
'a393',
'a394',
'a395',
'a396',
'a397',
'a398',
'a399',
'a400',
'a401',
'a402',
'a403',
'a404',
'a405',
'a406',
'a407',
'a408',
'a409',
'a410',
'a411',
'a412',
'a413',
'a414',
'a415',
'a416',
'a417',
'a418',
'a419',
'a420',
'a421',
'a422',
'a423',
'a424',
'a425',
'a426',
'a427',
'a428',
'a429',
'a430',
'a431',
'a432',
'a433',
'a434',
'a435',
'a436',
'a437',
'a438',
'a439',
'a440',
'a441',
'a442',
'a443',
'a444',
'a445',
'a446',
'a447',
'a448',
'a449',
'a450',
'a451',
'a452',
'a453',
'a454',
'a455',
'a456',
'a457',
'a458',

'a459',
'a460',
'a461',
'a462',
'a463',
'a464',
'a465',
'a466',
'a467',
'a468',
'a469',
'a470',
'a471',
'a472',
'a473',
'a474',
'a475',
'a476',
'a477',
'a478',
'a479',
'a480',
'a481',
'a482',
'a483',
'a484',
'a485',
'a486',
'a487',
'a488',
'a489',
'a490',
'a491',
'a492',
'a493',
'a494',
'a495',
'a496',
'a497',
'a498',
'a499',
'a500',
'a501',
'a502',
'a503',
'a504',
'a505',
'a506',
'a507',
'a508',
'a509',
'a510',
'a511',
'a512',
'a513',
'a514',
'a515',
'a516',
'a517',
'a518',
'a519',
'a520',
'a521',
'a522',
'a523',
'a524',
'a525',
'a526',
'a527',
'a528',
'a529',
'a530',

'a531',
'a532',
'a533',
'a534',
'a535',
'a536',
'a537',
'a538',
'a539',
'a540',
'a541',
'a542',
'a543',
'a544',
'a545',
'a546',
'a547',
'a548',
'a549',
'a550',
'a551',
'a552',
'a553',
'a554',
'a555',
'a556',
'a557',
'a558',
'a559',
'a560',
'a561',
'a562',
'a563',
'a564',
'a565',
'a566',
'a567',
'a568',
'a569',
'a570',
'a571',
'a572',
'a573',
'a574',
'a575',
'a576',
'a577',
'a578',
'a579',
'a580',
'a581',
'a582',
'a583',
'a584',
'a585',
'a586',
'a587',
'a588',
'a589',
'a590',
'a591',
'a592',
'a593',
'a594',
'a595',
'a596',
'a597',
'a598',
'a599',
'a600',
'a601',
'a602',

'a603',
'a604',
'a605',
'a606',
'a607',
'a608',
'a609',
'a610',
'a611',
'a612',
'a613',
'a614',
'a615',
'a616',
'a617',
'a618',
'a619',
'a620',
'a621',
'a622',
'a623',
'a624',
'a625',
'a626',
'a627',
'a628',
'a629',
'a630',
'a631',
'a632',
'a633',
'a634',
'a635',
'a636',
'a637',
'a638',
'a639',
'a640',
'a641',
'a642',
'a643',
'a644',
'a645',
'a646',
'a647',
'a648',
'a649',
'a650',
'a651',
'a652',
'a653',
'a654',
'a655',
'a656',
'a657',
'a658',
'a659',
'a660',
'a661',
'a662',
'a663',
'a664',
'a665',
'a666',
'a667',
'a668',
'a669',
'a670',
'a671',
'a672',
'a673',
'a674',

'a675',
'a676',
'a677',
'a678',
'a679',
'a680',
'a681',
'a682',
'a683',
'a684',
'a685',
'a686',
'a687',
'a688',
'a689',
'a690',
'a691',
'a692',
'a693',
'a694',
'a695',
'a696',
'a697',
'a698',
'a699',
'a700',
'a701',
'a702',
'a703',
'a704',
'a705',
'a706',
'a707',
'a708',
'a709',
'a710',
'a711',
'a712',
'a713',
'a714',
'a715',
'a716',
'a717',
'a718',
'a719',
'a720',
'a721',
'a722',
'a723',
'a724',
'a725',
'a726',
'a727',
'a728',
'a729',
'a730',
'a731',
'a732',
'a733',
'a734',
'a735',
'a736',
'a737',
'a738',
'a739',
'a740',
'a741',
'a742',
'a743',
'a744',
'a745',
'a746',

'a747',
'a748',
'a749',
'a750',
'a751',
'a752',
'a753',
'a754',
'a755',
'a756',
'a757',
'a758',
'a759',
'a760',
'a761',
'a762',
'a763',
'a764',
'a765',
'a766',
'a767',
'a768',
'a769',
'a770',
'a771',
'a772',
'a773',
'a774',
'a775',
'a776',
'a777',
'a778',
'a779',
'a780',
'a781',
'a782',
'a783',
'a784',
'a785',
'a786',
'a787',
'a788',
'a789',
'a790',
'a791',
'a792',
'a793',
'a794',
'a795',
'a796',
'a797',
'a798',
'a799',
'a800',
'a801',
'a802',
'a803',
'a804',
'a805',
'a806',
'a807',
'a808',
'a809',
'a810',
'a811',
'a812',
'a813',
'a814',
'a815',
'a816',
'a817',
'a818',

'a819',
'a820',
'a821',
'a822',
'a823',
'a824',
'a825',
'a826',
'a827',
'a828',
'a829',
'a830',
'a831',
'a832',
'a833',
'a834',
'a835',
'a836',
'a837',
'a838',
'a839',
'a840',
'a841',
'a842',
'a843',
'a844',
'a845',
'a846',
'a847',
'a848',
'a849',
'a850',
'a851',
'a852',
'a853',
'a854',
'a855',
'a856',
'a857',
'a858',
'a859',
'a860',
'a861',
'a862',
'a863',
'a864',
'a865',
'a866',
'a867',
'a868',
'a869',
'a870',
'a871',
'a872',
'a873',
'a874',
'a875',
'a876',
'a877',
'a878',
'a879',
'a880',
'a881',
'a882',
'a883',
'a884',
'a885',
'a886',
'a887',
'a888',
'a889',
'a890',

'a891',
'a892',
'a893',
'a894',
'a895',
'a896',
'a897',
'a898',
'a899',
'a900',
'a901',
'a902',
'a903',
'a904',
'a905',
'a906',
'a907',
'a908',
'a909',
'a910',
'a911',
'a912',
'a913',
'a914',
'a915',
'a916',
'a917',
'a918',
'a919',
'a920',
'a921',
'a922',
'a923',
'a924',
'a925',
'a926',
'a927',
'a928',
'a929',
'a930',
'a931',
'a932',
'a933',
'a934',
'a935',
'a936',
'a937',
'a938',
'a939',
'a940',
'a941',
'a942',
'a943',
'a944',
'a945',
'a946',
'a947',
'a948',
'a949',
'a950',
'a951',
'a952',
'a953',
'a954',
'a955',
'a956',
'a957',
'a958',
'a959',
'a960',
'a961',
'a962',

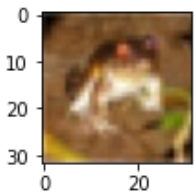
```
  'a963',
  'a964',
  'a965',
  'a966',
  'a967',
  'a968',
  'a969',
  'a970',
  'a971',
  'a972',
  'a973',
  'a974',
  'a975',
  'a976',
  'a977',
  'a978',
  'a979',
  'a980',
  'a981',
  'a982',
  'a983',
  'a984',
  'a985',
  'a986',
  'a987',
  'a988',
  'a989',
  'a990',
  'a991',
  'a992',
  'a993',
  'a994',
  'a995',
  'a996',
  'a997',
  'a998',
  'a999',
  ...],
 'target_names': ['class'],
 'DESCR': '**Author**: Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton    \n**Source**:
[University of Toronto](https://www.cs.toronto.edu/~kriz/cifar.html) - 2009  \n**Please c
ite**: Alex Krizhevsky (2009) Learning Multiple Layers of Features from Tiny Images, Tech
Report.\n\n**CIFAR-10 small**  \nThis is a 20,000 instance sample of the original CIFAR-1
0 dataset. Sampled randomly and stratified, with 2000 examples per class. Training and te
st set are merged. Find the corresponding task for the original train-test splits.\n\nCIF
AR-10 is a labeled subset of the [80 million tiny images dataset](http://groups.csail.mit
.edu/vision/TinyImages/). It (originally) consists 32x32 color images representing 10 cla
sses of objects:  \n0. airplane  \n1. automobile          \n2. bird          \n3. cat
\n4. deer          \n5. dog          \n6. frog          \n7. horse          \n8. ship
\n9. truck          \n\nThe classes are completely mutually exclusive. There is no overla
p between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort
. "Truck" includes only big trucks. Neither includes pickup trucks.\n\nThe original CIFAR
-10 dataset contains 6000 images per class. The original train-test split randomly divide
d these into 5000 train and 1000 test images per class.\n\n### Attribute description  \n\
nEach instance represents a 32x32 colour image as a 3072-value array. The first 1024 entr
ies contain the red channel values, the next 1024 the green, and the final 1024 the blue.
The image is stored in row-major order, so that the first 32 entries of the array are the
red channel values of the first row of the image.\n\nThe labels are encoded as integers i
n the range 0-9, corresponding to the numbered classes listed above\n\nDownloaded from op
enml.org.',
 'details': {'id': '40926',
  'name': 'CIFAR_10_small',
  'version': '1',
  'description_version': '1',
  'format': 'ARFF',
  'upload_date': '2017-09-26T23:24:38',
  'licence': 'Public',
  'url': 'https://api.openml.org/data/v1/download/16797612/CIFAR_10_small.arff',
  'parquet_url': 'http://openml1.win.tue.nl/dataset40926/dataset_40926.pq',
  'file_id': '16797612',
  'default_target_attribute': 'class',
  'tag': 'derived',
```

```
  'visibility': 'public',
  'minio_url': 'http://openml1.win.tue.nl/dataset40926/dataset_40926.pq',
  'status': 'active',
  'processing_date': '2018-10-04 07:18:53',
  'md5_checksum': '060250588efa7a126a72ab523c8f824c'},
 'url': 'https://www.openml.org/d/40926'}
```


In [3]:

```
#Figure out how to display some of the images in this data set, and display a couple.


#Plot Images
from matplotlib import pyplot as plt

index = 0
display_image = df.data
display_image = display_image.iloc[[index] , :]
#display_label = df.target
display_image = display_image.to_numpy().reshape(3,32,32).T /255
display_image = display_image.swapaxes(0,1)

from matplotlib import pyplot as plt
print(display_image.shape)
plt.figure(figsize=(1.5,1.5))
plt.imshow(display_image)

plt.show()
```

```
(32, 32, 3)
```



In [4]:

```
#train test split for x train y train x test y test on a 3/4 / 1/4 ratio

from sklearn.model_selection import train_test_split

X = df.data.to_numpy()
y = df.target.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=1/4, random_state=0)
```

In [5]:

```
from sklearn.preprocessing import StandardScaler

#Logistic Regression using multinomial.

#Stadard scaler as preprocessing step for solver sage

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

In [6]:

```
from sklearn.metrics import accuracy_score, log_loss
from sklearn.linear_model import LogisticRegression

#Logistic Regression using multinomial

#c_val=[x for x in np.arange(0.1,0.3,0.1)]
c_val=[0.1,0.2,0.4,0.8,1,1.2,1.5,1.8,2]
```

```
l1_res={}
l1_loss={}
#L1 regularization
for c in c_val:
    logisticRegression = LogisticRegression(solver='saga',multi_class='multinomial',C=c,
penalty='l1')
    logisticRegression.fit(X_train_scaled,y_train)
    #y_pred=logisticRegression.predict(X_test_scaled)
    l1_res[c]=logisticRegression.score(X_test_scaled,y_test)
    l1_loss[c]=log_loss(y_test,logisticRegression.predict_proba(X_test),labels=['0','1',
'2','3','4','5','6','7','8','9'])
```

```
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
```

In [7]:

```
#Report your training and test loss from above

lowest = max(l1_res, key=l1_res.get)


print("Highest accuracy was achieved for C value of: ", lowest," for an accuracy of: ",l1
_res[lowest])
print("loss values for each value of alpha: \n")

l1_loss
```

```
Highest accuracy was achieved for C value of:  0.1  for an accuracy of:  0.3924
loss values for each value of alpha:
```

Out[7]:

```
{0.1: 23.539285818688327,
 0.2: 23.363276415038847,
 0.4: 23.36344907095878,
 0.8: 23.394606103050187,
```

```
 1: 23.417009849507906,
 1.2: 23.42486491628771,
 1.5: 23.430558154939554,
 1.8: 23.44048803577338,
 2: 23.441695548492593}
```

In [8]:

```
l2_res={}
l2_loss={}
#L1 regularization
for c in c_val:
    logisticRegression = LogisticRegression(solver='saga',multi_class='multinomial',C=c,
penalty='l2')
    logisticRegression.fit(X_train_scaled,y_train)
    y_pred=logisticRegression.predict(X_test_scaled)
    l2_res[c]=logisticRegression.score(X_test_scaled,y_test)
    #l2_loss[c]=log_loss(y_test,y_pred)
    l2_loss[c]=log_loss(y_test,logisticRegression.predict_proba(X_test),labels=['0','1',
'2','3','4','5','6','7','8','9'])
```

```
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.p
y:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converg
e
  warnings.warn(
```

In [9]:

```
lowest = max(l2_res, key=l2_res.get)


print("Highest accuracy was achieved for C value of: ", lowest," for an accuracy of: ",l2
_res[lowest])

print("loss values for each value of alpha: \n")

l2_loss
```

```
Highest accuracy was achieved for C value of:  0.2  for an accuracy of:  0.3606
loss values for each value of alpha:
```

Out[9]:

```
{0.1: 23.45352142566835,
 0.2: 23.457818246438038,
 0.4: 23.46733723260369,
 0.8: 23.47359320242379,
 1: 23.474947828321564,
 1.2: 23.471798726310936,
 1.5: 23.47372546777759,
 1.8: 23.47541745069025,
 2: 23.47078899494649}
```

In [10]:

```
#How sparse can you make your solutions without deteriorating your testing error too much
?
#Here, we ask for a sparse solution that has test accuracy that is close to the best solu
tion you
#found.


l1_spar={}
l2_spar={}
#we also tested with original c vals and found that sparsity decreased the higher our c v
alues were
#for this part we lowered the c values further


c_val=[0.01,0.04,0.06,0.08]
# Set regularization parameter
for C in c_val:
    # turn down tolerance for short training time
    clf_l1_LR = LogisticRegression(C=C, penalty="l1", tol=0.01, solver="saga")
    clf_l2_LR = LogisticRegression(C=C, penalty="l2", tol=0.01, solver="saga")


    clf_l1_LR.fit(X_train_scaled,y_train)
    clf_l2_LR.fit(X_train_scaled,y_train)

    coef_l1_LR = clf_l1_LR.coef_.ravel()
    coef_l2_LR = clf_l2_LR.coef_.ravel()

    # coef_l1_LR contains zeros due to the
    # L1 sparsity inducing norm

    sparsity_l1_LR = np.mean(coef_l1_LR == 0) * 100
    sparsity_l2_LR = np.mean(coef_l2_LR == 0) * 100

    print("C=%.2f" % C)
    print("{:<40} {:.2f}%".format("Sparsity with L1 penalty:", sparsity_l1_LR))

    print("{:<40} {:.2f}%".format("Sparsity with L2 penalty:", sparsity_l2_LR))
    print("{:<40} {:.2f}".format("Score with L1 penalty:", clf_l1_LR.score(X_test_scaled
,y_test)))

    print("{:<40} {:.2f}".format("Score with L2 penalty:", clf_l2_LR.score(X_test_scaled
,y_test)))

    l1_spar[C]=sparsity_l1_LR
    l2_spar[C]=sparsity_l2_LR
```

```
C=0.01
Sparsity with L1 penalty:                95.99%
Sparsity with L2 penalty:                0.00%
Score with L1 penalty:                   0.38
Score with L2 penalty:                   0.39
C=0.04
Sparsity with L1 penalty:                88.00%
Sparsity with L2 penalty:                0.00%
Score with L1 penalty:                   0.40
```

```
Score with L2 penalty:                  0.37
C=0.06
Sparsity with L1 penalty:               81.25%
Sparsity with L2 penalty:               0.00%
Score with L1 penalty:                  0.40
Score with L2 penalty:                  0.37
C=0.08
Sparsity with L1 penalty:               73.36%
Sparsity with L2 penalty:               0.00%
Score with L1 penalty:                  0.40
Score with L2 penalty:                  0.37
```

Problem 2: MNIST multi class logistic regression¶

In [11]:

```
%%time

#Problem 2: Part 1: Use the fetch openml command from sklearn.datasets to import the MNIST data set,
from sklearn.datasets import fetch_openml
X, y = fetch_openml("mnist_784", version=1, return_X_y=True, as_frame=False)
```

```
CPU times: user 16.3 s, sys: 564 ms, total: 16.8 s
Wall time: 1min 53s
```

In [12]:

```
X.shape[0]
```

Out[12]:

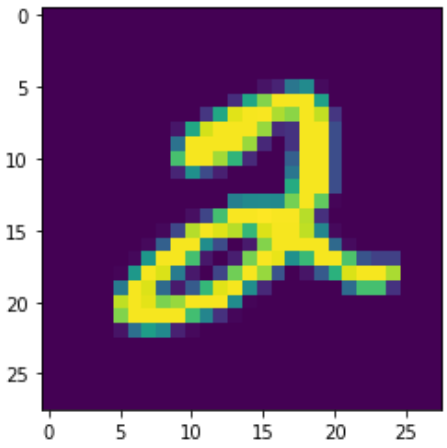
```
70000
```

In [13]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA

X_df = pd.DataFrame(X)
temp=X_df.iloc[5,:].values
temp = temp.reshape(28,28).astype('uint8')
plt.imshow(temp)
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7fb2e97943a0>
```



In [14]:

```
#Problem 2: Part 2: Choose a reasonable train-test split,
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=60000, test_size=10000
)
```

In [15]:

```
#Problem 2: Part 2: run multi class logistic regression

from sklearn.linear_model import LogisticRegression
mnist_lgr = LogisticRegression(C=3, multi_class='multinomial', penalty="l2", solver="saga", tol=0.1, max_iter=1000)
mnist_lgr.fit(X_train, y_train)
```

Out[15]:

```
LogisticRegression(C=3, max_iter=1000, multi_class='multinomial', solver='saga',
                   tol=0.1)
```

In [16]:

```
#y_pred for score calculation
y_pred = mnist_lgr.predict(X_test)
```

In [17]:

```
#Problem 2: Part 3: training and test loss from above
from sklearn.metrics import accuracy_score

score = (accuracy_score(y_test,y_pred))*100
print("Accuracy score is %.2f%%"% score)

sparsity = np.mean(mnist_lgr.coef_ == 0) * 100
print("Sparsity value for L2 is %.2f%%"% sparsity)
```

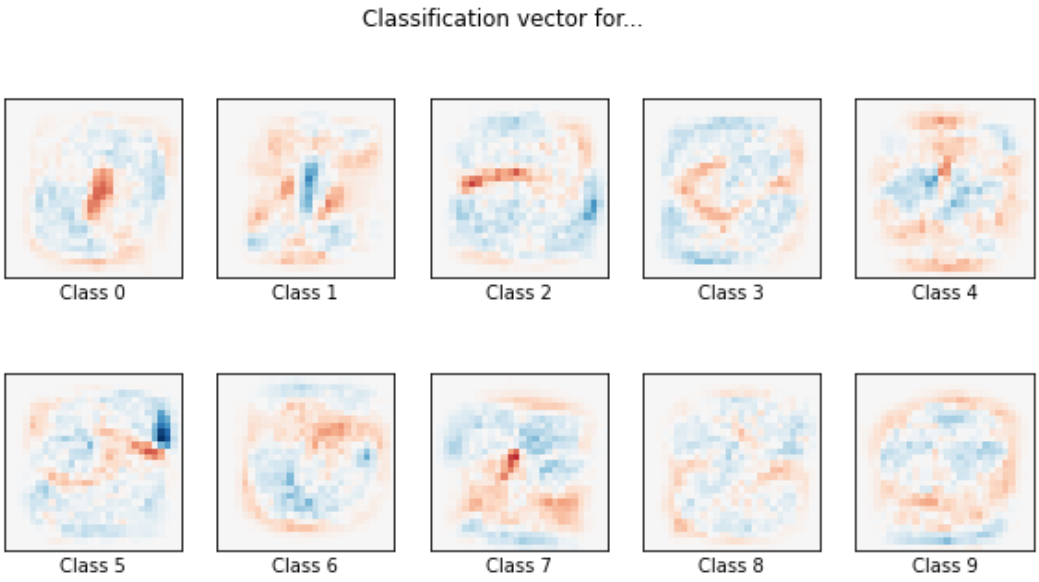
```
Accuracy score is 91.82%
Sparsity value for L2 is 8.55%
```

In [18]:

```
#Problem 2: Part 5: Note that in Logistic Regression, the coefficients returned (i.e., the β's) are the same dimen- sion as the data. Therefore we#
# can pretend that
# the coefficients of the solution are an image of the same dimension, and plot it. Do this for the 10 sets of coefficients that correspond to the
# 10 classes. You should observe that, at least for the sparse solutions, these "kind of" look like the digits they are classifying
import matplotlib.pyplot as plt

coef = mnist_lgr.coef_.copy()

plt.figure(figsize=(10, 5))

scale = np.abs(coef).max()
for i in range(10):
    l1_plot = plt.subplot(2, 5, i + 1)
    l1_plot.imshow(
        coef[i].reshape(28, 28),
        interpolation="nearest",
        cmap=plt.cm.RdBu,
        vmin=-scale,
        vmax=scale,
    )
    l1_plot.set_xticks(())
    l1_plot.set_yticks(())
    l1_plot.set_xlabel("Class %i" % i)
plt.suptitle("Classification vector for...")
```

Out[18]:

```
Text(0.5, 0.98, 'Classification vector for...')
```



In [19]:

```
#Problem 2: part 2: Cross entropy loss

from sklearn.metrics import log_loss

cross_entropy_loss = log_loss(y_test, mnist_lgr.predict_proba(X_test))
print("Logistic regression cross entropy loss: ",cross_entropy_loss)
```

```
Logistic regression cross entropy loss:  0.29243672509175805
```

In [20]:

```
#Problem 2: part 4: Choose an ℓ1 regularizer (penalty), and see if you can get a sparse s
olution with almost as
#good accuracy

mnist_lgr_l1 = LogisticRegression(C=3, multi_class='multinomial', penalty="l1", solver="
saga", tol=0.1)
mnist_lgr_l1.fit(X_train, y_train)

y_pred_l1 = mnist_lgr_l1.predict(X_test)

score_l1 = accuracy_score(y_test,y_pred_l1)
print("Accuracy score is ", score_l1)

sparsity_l1 = np.mean(mnist_lgr_l1.coef_ == 0) * 100
print("Sparsity value for L1 is ", sparsity_l1)
```

```
Accuracy score is  0.9178
Sparsity value for L1 is  13.380102040816327
```

#Problem 3: part 1: Use Random Forests to try to get the best possible test accuracy on MNIST. Use Cross

Validation to find the best settings.

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

rf_mnist = RandomForestClassifier(n_estimators=150)
rf_mnist.fit(X_train,y_train)
```

Out[21]:

```
▾ RandomForestClassifier
```

```
RandomForestClassifier(n_estimators=150)
```

In [22]:

```
#Cross Validation Score for Random Forest Classifier
pred=rf_mnist.predict(X_test)

score = cross_val_score(rf_mnist, X_train, y_train)
print ("Cross Validation Score for Random Forest Classifier is: ",np.mean(score)*100)
```

```
Cross Validation Score for Random Forest Classifier is:  96.75333333333333
```

In [23]:

```
#Problem 3: part 2: use Gradient Boosting to do the same (we used catboost classifier) an
d hyperparams for best model

import catboost as ctb


#hyperparameters for best catboost model
cat = ctb.CatBoostClassifier(iterations=1000,
                 learning_rate=0.03, depth=6, loss_function='MultiClass',
        early_stopping_rounds= 1000, verbose=0)


cat.fit(X_train, y_train)
```

```
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/xgboost/compat.py:36: Futur
eWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future ver
sion. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

Out[23]:

```
<catboost.core.CatBoostClassifier at 0x7fb2cabbb910>
```

In [24]:

```
y_pred = cat.predict(X_test)
```

In [25]:

```
#catboost accuracy score
score = (accuracy_score(y_test,y_pred))*100
print("Accuracy score is %.2f%%"% score)
```

```
Accuracy score is 96.74%
```

In [26]:

```
#cross validation score
cv_results = cross_val_score(cat, X_train, y_train,
                  cv = 2, scoring='accuracy', n_jobs = -1, verbose=0)
```

```
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/xgboost/compat.py:36: Futur
eWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future ver
sion. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
/Users/manvimahajan/opt/anaconda3/lib/python3.9/site-packages/xgboost/compat.py:36: Futur
eWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future ver
sion. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [27]:

```
print ("Cross Validation score for CatBoost Classifier: ",np.mean(cv_results)*100)
```

```
Cross Validation score for CatBoost Classifier:  95.95333333333333
```

We got a sparsity of 88% with highest accuracy of 40% as our best result at c=0.04 that beats our previous accuracy of 39% at c=0.1

#Problem 4 - Part 1 - CIFAR : What is the best accuracy you can get on the test data, by tuning Random Forests?

In a separate jupyter notebook

In [ ]:

Problem 4 - CIFAR Continued

In [17]:

```
#Problem 4 - Part 1 - CIFAR : What is the best accuracy you can get on the test data, by
tuning Random Forests?

from sklearn.ensemble import RandomForestClassifier

df = fetch_openml('CIFAR_10')
X = df.data.to_numpy()
y = df.target.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=1/4, random_state=0)


rf_cifar = RandomForestClassifier(n_estimators=150)
rf_cifar.fit(X_train, y_train)
```

Out[17]:

```
▾          RandomForestClassifier
RandomForestClassifier(n_estimators=150)
```

In [18]:

```
from sklearn.model_selection import cross_val_score

pred=rf_cifar.predict(X_test)

score = cross_val_score(rf_cifar, X_train, y_train)
print ("Cross Validation Score for Random Forest Classifier is: ",np.mean(score)*100)
```

```
Cross Validation Score for Random Forest Classifier is:  46.355555555555554
```

In [19]:

```
#Problem 4 - Part 2 - What is the best accuracy you can get on the test data, by tuning a
ny model including Gradient boosting?

import catboost as ctb

#hyperparameters for best catboost model
cat = ctb.CatBoostClassifier(iterations=1000,
                             learning_rate=0.03, depth=6, loss_function='MultiClass',
                             early_stopping_rounds= 1000, verbose=0)


cat.fit(X_train, y_train)
```

Out[19]:

```
<catboost.core.CatBoostClassifier at 0x3859bf970>
```

In [20]:

```
y_pred = cat.predict(X_test)
#catboost accuracy score
score = (accuracy_score(y_test, y_pred)) * 100
print("Accuracy score is %.2f%%" % score)
#cross validation score
cv_results = cross_val_score(cat, X_train, y_train,
                             cv=2, scoring='accuracy', n_jobs=-1, verbose=0)

print("Cross Validation score for CatBoost Classifier: ", np.mean(cv_results) * 100)
```

```
Accuracy score is 51.73%
```

```
Cross Validation score for CatBoost Classifier:  50.38222222222222
```

In [ ]:

```
```