Python Lab-8--MNIST Exr 1

Name- Jatin Karthik

Matriculation no.-313301

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1. Loading the MNIST digits dataset via sklearn provided built-in utility function(s).

In [5]:
```
from sklearn.datasets import load_digits
#digits = load_digits()
#print(digits.data.shape)


mnist = load_digits()
type(mnist)

mnist.keys()
```

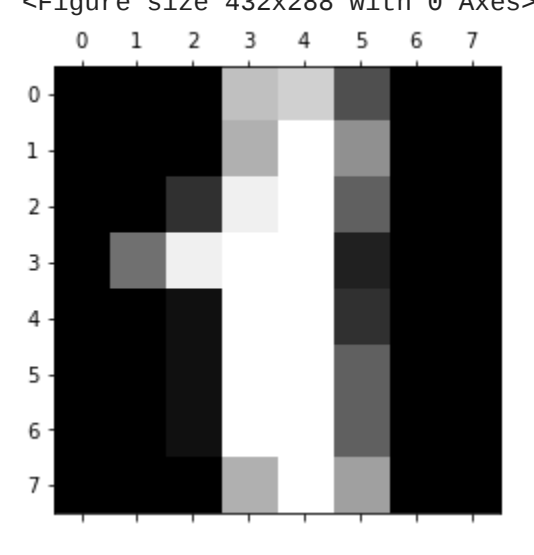
Out[5]:
```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

In [14]:
```
X=pd.DataFrame(mnist.data)


import matplotlib.pyplot as plt
plt.gray()
plt.matshow(digits.images[1])

plt.show()
X
```

```
<Figure size 432x288 with 0 Axes>
```



Out[14]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0

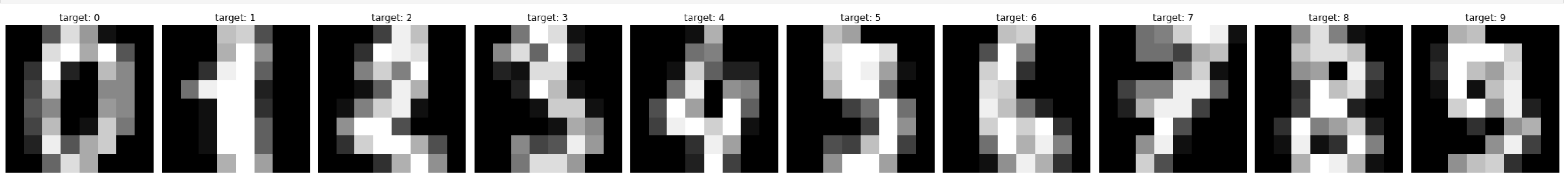
1797 rows × 64 columns

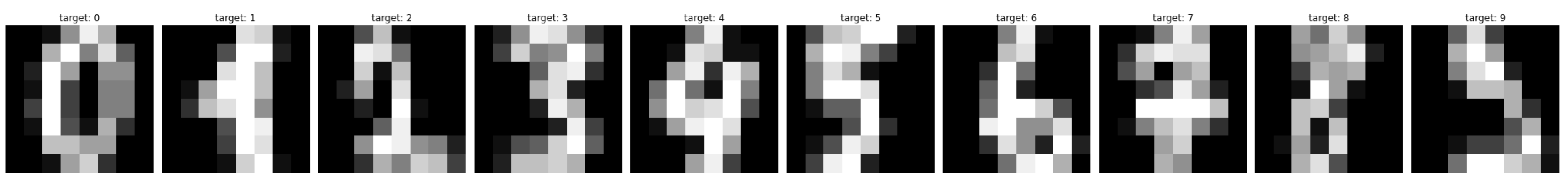
In [33]:
```
Y=pd.DataFrame(mnist.target)
Y=np.ravel(Y)
Y
```

Out[33]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [21]:
```
fig, axes = plt.subplots(2, 10, figsize=(28,28))
for i in range(20):
    axes[i//10, i %10].imshow(mnist.images[i], cmap='gray');
    axes[i//10, i %10].axis('off')
    axes[i//10, i %10].set_title(f"target: {mnist.target[i]}")

plt.tight_layout()
```





In [24]:
```
np.shape(mnist.images)
```

Out[24]:
```
(1797, 8, 8)
```

As we can see there are 1797 images in toatl with 8x8 pixels, so we need to set aside 20% of the images for testing which would be approximately 360 images for testing and 1437 images for training.

In [27]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2)
```

MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.

In [80]:
```
from sklearn.model_selection import cross_val_score
#Importing MLPClassifier
from sklearn.neural_network import MLPClassifier

#Initializing the MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(150,50), max_iter=50,activation = 'relu',solver='adam',random_state=3116)
```

In [81]:
```
#Fitting the training data to the network
m=classifier.fit(X_train, Y_train)
m
```

Out[81]:
```
MLPClassifier(hidden_layer_sizes=(150, 50), max_iter=50, random_state=3116)
```

In [82]:
```
#Predicting y for X_val
y_pred = classifier.predict(X_test)
```

Accuracy without cross validation

In [83]:
```
#Importing Confusion Matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#Comparing the predictions against the actual observations in y_val
cm= confusion_matrix(y_pred, Y_test)

#Printing the accuracy
print("Accuracy of MLPClassifier : ", cm)
accuracy_score(Y_test, classifier.predict(X_test))
```

```
Accuracy of MLPClassifier :  [[44  0  0  0  0  0  0  0  0  0]
 [ 0 37  0  0  0  0  0  0  0  0]
 [ 0  0 38  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  0]
 [ 0  0  0  0 37  0  0  0  0  0]
 [ 0  0  0  0  0 33  0  0  0  1]
 [ 0  0  0  0  0  0 38  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  0]
 [ 0  0  0  0  0  0  0  0 34  0]
 [ 0  0  0  2  0  0  0  0  0 27]]
```

Out[83]:
```
0.9916666666666667
```

With k cross validation

In [84]:
```
import warnings
warnings.filterwarnings('ignore')


scores=cross_val_score(m,X_train,Y_train,scoring='r2',cv=5)
print(scores)
print("Average scores with 5 cross validation on train dataset ", np.mean(scores))
```

```
[0.97323037 0.93538293 0.88156867 0.94299816 0.90430782]
Average scores with 5 cross validation on train dataset  0.9274975882600204
```

In [85]:
```
from sklearn.model_selection import cross_val_predict
pred=cross_val_predict(m,X_test,Y_test)
scores_test=cross_val_score(m,X_test,Y_test,cv=5)
print("Accuracy on test datset is", np.mean(scores_test))
```

```
Accuracy on test datset is 0.9361111111111111
```

Random search hyperparameter optimisation:- MLPClassifier has the hyperparameters--activation, alpha, batch_size, beta_1,beta_2 early_stopping, epsilon,hidden_layer_sizes,learning_rate,learning_rate_init, max_iter, momentum,nesterovs_momentum, power_t, random_state,shuffle, solver, tol, validation_fraction,verbose and warm_start. In my opinion the three things which really matter in these hyperparameters are alpha, hidden_layer_sizes and iterations(which are epochs). So, I decided to define a random search function for these.

In [104…
```
#try using random function and implement this

def randomsearch():
    i = 0
    df = pd.DataFrame(columns = ['alpha','max_iter','hidden_layer_sizes','train_acc','test_acc'])
    for a in [0.00001,0.0001,0.001,0.01, 0.1, 1, 10]:
        for mi in [10,100,200,500,1000,2000]:
            for l in [(10,5),(50,20),(100,50),(150,60),(200,100)]:

            #st = time()
                mlp = MLPClassifier(alpha=a,hidden_layer_sizes=l,max_iter=mi)
                mlp.fit(X_train, Y_train)
            #end = time() - st

                acc_tr = accuracy_score(Y_train, mlp.predict(X_train)) # Train Accuracy
                acc = accuracy_score(Y_test, mlp.predict(X_test)) # Test Accuracy
                df.loc[i] = [a,mi,l,acc_tr,acc]
                i=i+1
    return df
```

In [105…
```
table=randomsearch()
```

In [110…
```
#Displayed as a string rather than a dataframe so that the whole table is visible
print(table.to_string())
```

	alpha	max_iter	hidden_layer_sizes	train_acc	test_acc
0	0.00001	10	(10, 5)	0.165623	0.166667
1	0.00001	10	(50, 20)	0.729297	0.705556
2	0.00001	10	(100, 50)	0.954071	0.952778
3	0.00001	10	(150, 60)	0.969381	0.977778
4	0.00001	10	(200, 100)	0.990953	0.980556
5	0.00001	100	(10, 5)	0.892136	0.877778
6	0.00001	100	(50, 20)	1.000000	0.972222
7	0.00001	100	(100, 50)	1.000000	0.986111
8	0.00001	100	(150, 60)	1.000000	0.986111
9	0.00001	100	(200, 100)	1.000000	0.991667
10	0.00001	200	(10, 5)	0.949200	0.944444
11	0.00001	200	(50, 20)	1.000000	0.983333
12	0.00001	200	(100, 50)	1.000000	0.986111
13	0.00001	200	(150, 60)	1.000000	0.991667
14	0.00001	200	(200, 100)	1.000000	0.988889
15	0.00001	500	(10, 5)	0.988866	0.958333
16	0.00001	500	(50, 20)	1.000000	0.975000
17	0.00001	500	(100, 50)	1.000000	0.988889
18	0.00001	500	(150, 60)	1.000000	0.986111
19	0.00001	500	(200, 100)	1.000000	0.991667
20	0.00001	1000	(10, 5)	0.999304	0.947222
21	0.00001	1000	(50, 20)	1.000000	0.991667
22	0.00001	1000	(100, 50)	1.000000	0.994444
23	0.00001	1000	(150, 60)	1.000000	0.994444
24	0.00001	1000	(200, 100)	1.000000	0.988889
25	0.00001	2000	(10, 5)	0.999304	0.933333
26	0.00001	2000	(50, 20)	1.000000	0.988889
27	0.00001	2000	(100, 50)	1.000000	0.983333
28	0.00001	2000	(150, 60)	1.000000	0.980556
29	0.00001	2000	(200, 100)	1.000000	0.994444
30	0.00010	10	(10, 5)	0.117606	0.133333
31	0.00010	10	(50, 20)	0.745303	0.730556
32	0.00010	10	(100, 50)	0.923452	0.919444
33	0.00010	10	(150, 60)	0.971468	0.972222
34	0.00010	10	(200, 100)	0.992345	0.975000
35	0.00010	100	(10, 5)	0.776618	0.769444
36	0.00010	100	(50, 20)	0.999304	0.980556
37	0.00010	100	(100, 50)	1.000000	0.972222
38	0.00010	100	(150, 60)	1.000000	0.994444
39	0.00010	100	(200, 100)	1.000000	0.997222
40	0.00010	200	(10, 5)	0.933890	0.900000
41	0.00010	200	(50, 20)	1.000000	0.983333
42	0.00010	200	(100, 50)	1.000000	0.986111
43	0.00010	200	(150, 60)	1.000000	0.983333
44	0.00010	200	(200, 100)	1.000000	0.983333
45	0.00010	500	(10, 5)	0.976340	0.927778
46	0.00010	500	(50, 20)	1.000000	0.983333
47	0.00010	500	(100, 50)	1.000000	0.991667
48	0.00010	500	(150, 60)	1.000000	0.983333
49	0.00010	500	(200, 100)	1.000000	0.983333
50	0.00010	1000	(10, 5)	1.000000	0.941667
51	0.00010	1000	(50, 20)	1.000000	0.975000
52	0.00010	1000	(100, 50)	1.000000	0.975000
53	0.00010	1000	(150, 60)	1.000000	0.988889
54	0.00010	1000	(200, 100)	1.000000	0.991667
55	0.00010	2000	(10, 5)	0.999304	0.952778
56	0.00010	2000	(50, 20)	1.000000	0.980556
57	0.00010	2000	(100, 50)	1.000000	0.994444
58	0.00010	2000	(150, 60)	1.000000	0.994444
59	0.00010	2000	(200, 100)	1.000000	0.994444
60	0.00100	10	(10, 5)	0.176061	0.191667
61	0.00100	10	(50, 20)	0.688935	0.700000
62	0.00100	10	(100, 50)	0.963118	0.969444
63	0.00100	10	(150, 60)	0.979819	0.975000
64	0.00100	10	(200, 100)	0.995129	0.983333
65	0.00100	100	(10, 5)	0.859429	0.841667
66	0.00100	100	(50, 20)	1.000000	0.986111
67	0.00100	100	(100, 50)	1.000000	0.986111
68	0.00100	100	(150, 60)	1.000000	0.991667
69	0.00100	100	(200, 100)	1.000000	0.988889
70	0.00100	200	(10, 5)	0.929019	0.900000
71	0.00100	200	(50, 20)	1.000000	0.986111
72	0.00100	200	(100, 50)	1.000000	0.980556
73	0.00100	200	(150, 60)	1.000000	0.988889
74	0.00100	200	(200, 100)	1.000000	0.991667
75	0.00100	500	(10, 5)	0.993737	0.930556
76	0.00100	500	(50, 20)	1.000000	0.986111
77	0.00100	500	(100, 50)	1.000000	0.988889
78	0.00100	500	(150, 60)	1.000000	0.988889
79	0.00100	500	(200, 100)	1.000000	0.988889
80	0.00100	1000	(10, 5)	0.999304	0.925000
81	0.00100	1000	(50, 20)	1.000000	0.986111
82	0.00100	1000	(100, 50)	1.000000	0.983333
83	0.00100	1000	(150, 60)	1.000000	0.994444
84	0.00100	1000	(200, 100)	1.000000	0.988889
85	0.00100	2000	(10, 5)	0.999304	0.944444
86	0.00100	2000	(50, 20)	1.000000	0.983333
87	0.00100	2000	(100, 50)	1.000000	0.991667
88	0.00100	2000	(150, 60)	1.000000	0.986111
89	0.00100	2000	(200, 100)	1.000000	0.988889
90	0.01000	10	(10, 5)	0.100905	0.091667
91	0.01000	10	(50, 20)	0.729993	0.708333
92	0.01000	10	(100, 50)	0.943633	0.936111
93	0.01000	10	(150, 60)	0.972860	0.944444
94	0.01000	10	(200, 100)	0.989562	0.983333
95	0.01000	100	(10, 5)	0.843424	0.869444
96	0.01000	100	(50, 20)	1.000000	0.969444
97	0.01000	100	(100, 50)	1.000000	0.991667
98	0.01000	100	(150, 60)	1.000000	0.975000
99	0.01000	100	(200, 100)	1.000000	0.988889
100	0.01000	200	(10, 5)	0.949896	0.908333
101	0.01000	200	(50, 20)	1.000000	0.983333
102	0.01000	200	(100, 50)	1.000000	0.983333
103	0.01000	200	(150, 60)	1.000000	0.986111
104	0.01000	200	(200, 100)	1.000000	0.988889
105	0.01000	500	(10, 5)	0.993041	0.930556
106	0.01000	500	(50, 20)	1.000000	0.975000
107	0.01000	500	(100, 50)	1.000000	0.988889
108	0.01000	500	(150, 60)	1.000000	0.986111
109	0.01000	500	(200, 100)	1.000000	0.988889
110	0.01000	1000	(10, 5)	1.000000	0.925000
111	0.01000	1000	(50, 20)	1.000000	0.977778
112	0.01000	1000	(100, 50)	1.000000	0.991667
113	0.01000	1000	(150, 60)	1.000000	0.991667
114	0.01000	1000	(200, 100)	1.000000	0.991667
115	0.01000	2000	(10, 5)	0.986778	0.927778
116	0.01000	2000	(50, 20)	1.000000	0.977778
117	0.01000	2000	(100, 50)	1.000000	0.991667
118	0.01000	2000	(150, 60)	1.000000	0.991667
119	0.01000	2000	(200, 100)	1.000000	0.983333
120	0.10000	10	(10, 5)	0.172582	0.150000
121	0.10000	10	(50, 20)	0.803758	0.802778
122	0.10000	10	(100, 50)	0.951983	0.947222
123	0.10000	10	(150, 60)	0.979123	0.983333
124	0.10000	10	(200, 100)	0.986778	0.977778
125	0.10000	100	(10, 5)	0.520529	0.544444
126	0.10000	100	(50, 20)	1.000000	0.975000
127	0.10000	100	(100, 50)	1.000000	0.988889
128	0.10000	100	(150, 60)	1.000000	0.988889
129	0.10000	100	(200, 100)	1.000000	0.986111
130	0.10000	200	(10, 5)	0.967989	0.922222
131	0.10000	200	(50, 20)	1.000000	0.980556
132	0.10000	200	(100, 50)	1.000000	0.988889
133	0.10000	200	(150, 60)	1.000000	0.988889
134	0.10000	200	(200, 100)	1.000000	0.988889
135	0.10000	500	(10, 5)	0.984690	0.916667
136	0.10000	500	(50, 20)	1.000000	0.975000
137	0.10000	500	(100, 50)	1.000000	0.983333
138	0.10000	500	(150, 60)	1.000000	0.983333
139	0.10000	500	(200, 100)	1.000000	0.983333
140	0.10000	1000	(10, 5)	0.998608	0.927778
141	0.10000	1000	(50, 20)	1.000000	0.983333
142	0.10000	1000	(100, 50)	1.000000	0.980556
143	0.10000	1000	(150, 60)	1.000000	0.991667
144	0.10000	1000	(200, 100)	1.000000	0.994444
145	0.10000	2000	(10, 5)	0.999304	0.947222
146	0.10000	2000	(50, 20)	1.000000	0.983333
147	0.10000	2000	(100, 50)	1.000000	0.988889
148	0.10000	2000	(150, 60)	1.000000	0.991667
149	0.10000	2000	(200, 100)	1.000000	0.988889
150	1.00000	10	(10, 5)	0.225470	0.244444
151	1.00000	10	(50, 20)	0.795407	0.791667
152	1.00000	10	(100, 50)	0.950592	0.955556
153	1.00000	10	(150, 60)	0.977035	0.963889
154	1.00000	10	(200, 100)	0.988866	0.980556
155	1.00000	100	(10, 5)	0.835073	0.791667
156	1.00000	100	(50, 20)	0.999304	0.983333
157	1.00000	100	(100, 50)	1.000000	0.988889
158	1.00000	100	(150, 60)	1.000000	0.986111
159	1.00000	100	(200, 100)	1.000000	0.991667
160	1.00000	200	(10, 5)	0.935282	0.902778
161	1.00000	200	(50, 20)	1.000000	0.977778
162	1.00000	200	(100, 50)	1.000000	0.991667
163	1.00000	200	(150, 60)	1.000000	0.991667
164	1.00000	200	(200, 100)	0.999304	0.991667
165	1.00000	500	(10, 5)	0.993737	0.944444
166	1.00000	500	(50, 20)	1.000000	0.986111
167	1.00000	500	(100, 50)	1.000000	0.988889
168	1.00000	500	(150, 60)	1.000000	0.986111
169	1.00000	500	(200, 100)	0.998608	0.983333
170	1.00000	1000	(10, 5)	0.998608	0.977778
171	1.00000	1000	(50, 20)	1.000000	0.983333
172	1.00000	1000	(100, 50)	1.000000	0.986111
173	1.00000	1000	(150, 60)	1.000000	0.983333
174	1.00000	1000	(200, 100)	1.000000	0.991667
175	1.00000	2000	(10, 5)	0.997216	0.969444
176	1.00000	2000	(50, 20)	1.000000	0.988889
177	1.00000	2000	(100, 50)	1.000000	0.986111
178	1.00000	2000	(150, 60)	1.000000	0.991667
179	1.00000	2000	(200, 100)	1.000000	0.991667
180	10.00000	10	(10, 5)	0.270007	0.275000
181	10.00000	10	(50, 20)	0.716075	0.705556
182	10.00000	10	(100, 50)	0.956855	0.955556
183	10.00000	10	(150, 60)	0.970772	0.975000
184	10.00000	10	(200, 100)	0.985386	0.969444
185	10.00000	100	(10, 5)	0.831594	0.827778
186	10.00000	100	(50, 20)	0.981211	0.983333
187	10.00000	100	(100, 50)	0.982603	0.983333
188	10.00000	100	(150, 60)	0.982603	0.983333
189	10.00000	100	(200, 100)	0.972164	0.977778
190	10.00000	200	(10, 5)	0.872651	0.852778
191	10.00000	200	(50, 20)	0.979123	0.972222
192	10.00000	200	(100, 50)	0.978427	0.983333
193	10.00000	200	(150, 60)	0.980515	0.980556
194	10.00000	200	(200, 100)	0.979819	0.980556
195	10.00000	500	(10, 5)	0.969381	0.958333
196	10.00000	500	(50, 20)	0.981907	0.977778
197	10.00000	500	(100, 50)	0.981211	0.975000
198	10.00000	500	(150, 60)	0.981907	0.980556
199	10.00000	500	(200, 100)	0.974252	0.977778
200	10.00000	1000	(10, 5)	0.969381	0.963889
201	10.00000	1000	(50, 20)	0.980515	0.980556
202	10.00000	1000	(100, 50)	0.984690	0.980556
203	10.00000	1000	(150, 60)	0.978427	0.980556
204	10.00000	1000	(200, 100)	0.977731	0.972222
205	10.00000	2000	(10, 5)	0.952679	0.961111
206	10.00000	2000	(50, 20)	0.981907	0.977778
207	10.00000	2000	(100, 50)	0.979819	0.975000
208	10.00000	2000	(150, 60)	0.981907	0.980556
209	10.00000	2000	(200, 100)	0.980515	0.977778

In [115…
```
table.max()
```

Out[115…
```
alpha                        10
max_iter                   2000
hidden_layer_sizes    (200, 100)
train_acc                     1
test_acc               0.997222
dtype: object
```

Inference:--The more the hidden_layer_sizes with the same minimal lpha and iterations, the better the test accuracy becomes, so all we need to find is a sweet spot between alpha, iterations and a larger hidden_layer_sizes to give the best test accuracy.

Therefore,The best hyperparameter values are when alpha=10, max_iter=2000 and hidden_layer_sizes=(200,100)

In [ ]: