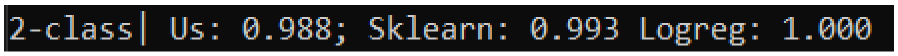
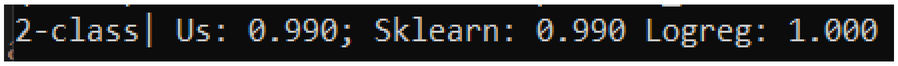
a.**I am going with the digit classes 3 and 6 as they are dissimilar.
X, y = load_digits(classes=(3, 6), n=10000)
hidden_layer_sizes = (8,)
max_iter = 200



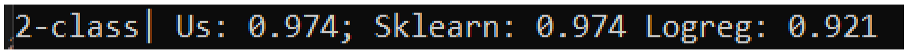
```
2-class| Us: 0.988; Sklearn: 0.993 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (8,)
max_iter = 200



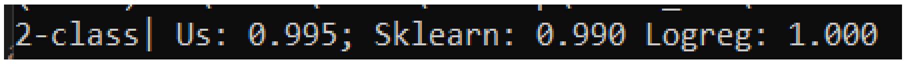
```
2-class| Us: 0.990; Sklearn: 0.990 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=1000)
hidden_layer_sizes = (8,)
max_iter = 200



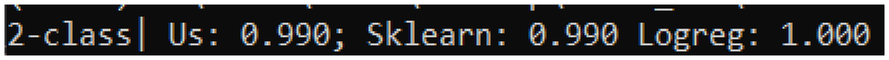
```
2-class| Us: 0.974; Sklearn: 0.974 Logreg: 0.921
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (16,)
max_iter = 200



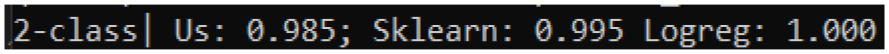
```
2-class| Us: 0.995; Sklearn: 0.990 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (8, 8)
max_iter = 200



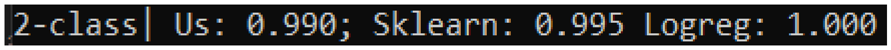
```
2-class| Us: 0.990; Sklearn: 0.990 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (16, 16)
max_iter = 200



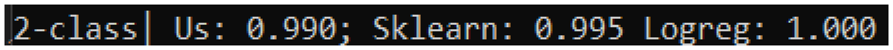
```
2-class| Us: 0.985; Sklearn: 0.995 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (16, 8)
max_iter = 200



```
2-class| Us: 0.990; Sklearn: 0.995 Logreg: 1.000
```

X, y = load_digits(classes=(3, 6), n=5000)
hidden_layer_sizes = (16, 8)
max_iter = 400



```
2-class| Us: 0.990; Sklearn: 0.995 Logreg: 1.000
```

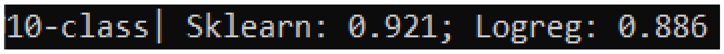
From the parameters altered:

- “classes” simply uses two classes of digits (0-9) for the binary classification.
- “n” controls the number of examples to use for training and testing.
- “hidden_layer_sizes” is a tuple that specifies the number of nodes in each hidden layer.
- “max_iter” specifies the maximum number of iterations (epochs) for training the model.

Of course, for each class combination used, will differ in the amount that each parameter optimizes the accuracy, especially if the class pair are dissimilar to one another (easier to distinguish a digit between 5 and 1, rather than 7 and 1). It can be seen that if one increases ‘n’, the example set size, generally the accuracy increases, however there is a tradeoff between accuracy and training time. Yet, the keyword is generally, as seen in the experiments above, having a large example set size may not yield higher accuracy, but a high *enough* example set size may optimize the accuracy. This was a surprising result. As for “hidden_layer_sizes”, it is less clear when trying out different combinations of layers. It seems that having a *high enough* amount of nodes for a given layer can optimize the accuracy as having too many nodes and too many layers can lead to overfitting and thus lower accuracy. The parameter, “max_iter”, has a seemingly minimal effect. Upon increasing the “max_iter” the accuracy can improve slightly if the model hasn’t converged, otherwise, this parameter will not have much of an effect on the accuracy.

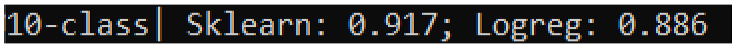
b.

hidden_layer_sizes = (32,)
max_iter = 200
sklearn_kwargs = {}
X, y = load_digits(classes=range(10), n=10000)



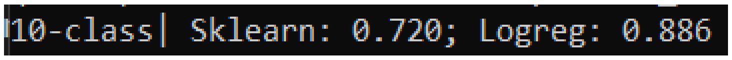
```
10-class| Sklearn: 0.921; Logreg: 0.886
```

hidden_layer_sizes = (32,)
max_iter = 200
sklearn_kwargs = {‘alpha’: 0.1}
X, y = load_digits(classes=range(10), n=10000)



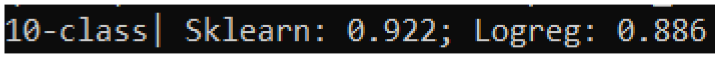
```
10-class| Sklearn: 0.917; Logreg: 0.886
```

hidden_layer_sizes = (32,)
max_iter = 100
sklearn_kwargs = {‘learning_rate_init’: 0.01}
X, y = load_digits(classes=range(10), n=10000)



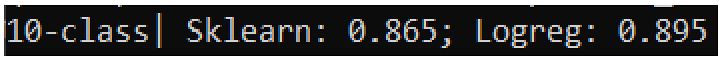
```
10-class| Sklearn: 0.720; Logreg: 0.886
```

hidden_layer_sizes = (64, 32)
max_iter = 100
sklearn_kwargs = {‘alpha’: 0.1}
X, y = load_digits(classes=range(10), n=10000)



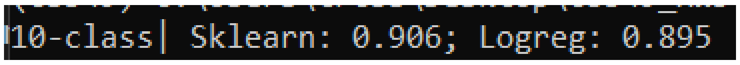
```
10-class| Sklearn: 0.922; Logreg: 0.886
```

hidden_layer_sizes = (64,32)
max_iter = 100
sklearn_kwargs = {‘solver’: ‘adam’}
X, y = load_digits(classes=range(10), n=10000)



```
10-class| Sklearn: 0.865; Logreg: 0.895
```

hidden_layer_sizes = (32,)
max_iter = 100
sklearn_kwargs = {‘solver’: ’adam’}
X, y = load_digits(classes=range(10), n=10000)



```
10-class| Sklearn: 0.906; Logreg: 0.895
```

The addition of increasing layers, along with nodes, helps accuracy up to a certain point, at which a surplus of layers can yield a decreasing performance. However, in this case having multiple layers and nodes yielded the best accuracy, given we are now considering all ten classes. The best accuracy was the combination of 64 nodes on layer 1 and 32 nodes on layer 2, along with a large example size and alpha = 0.01. This makes sense since we are now considering more digits to classify rather than just between two. I used a couple of sklearn parameters to see how they influence the accuracy of the model, it seems that in my experimentation, using the regularization parameter alpha worked best, to prevent overfitting of the multiple layers that are used for training.

c.

The current implementation of the Model class will not work for ten-class classification because it only has one output neuron, which is suitable for binary classification. For multi-class classification, the neural network should have as many output neurons as the number of classes. In order for the Model class to handle ten-class classification, one would need to change the loss function and the output layer. One can use the categorical cross-entropy loss function, which is appropriate for multi-class classification problems, and one would need to change the number of output neurons in the output layer to ten to allow the Model class to be more well-suited for this application.