This task is to build a classification model for MNIST dataset. For this experiment, we use 60,000 28x28 grayscale images of the 10 digits for train data and 10,000 images for test data, as can be seen from the following figure.

```
# Load data from https://www.openml.org/d/554
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

```
Automatically created module for IPython interactive environment
```

```
X = X / 255.

# rescale the data, use the traditional train/test split
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]

print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

The above figure is data preparation for scikit-learn, while the following figure shows data preparation for keras.

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
60000 train samples
10000 test samples
```

In order to build neural network model with scikit-learn for this problem, the following line of code is used.

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, alpha=1e-4,
                    solver='adam', verbose=20, random_state=1,
                    learning_rate_init=.1)
```

While for keras, we can use the following code.

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

As can be seen from the two figures, I used relu as the activation function. For scikit-learn, 1 hidden layer with 50 neurons was used while for keras, 2 hidden layers with 512 neurons each were used. For the optimizer, adam was used. For both scikit-learn and keras, after building the model, we can call 'fit' function to fit the model to the training data. The figure bellow shows the syntax for scikit learn,

```
mlp.fit(X_train,y_train)
```

While the following figure show the syntax for keras.

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=2,
                    validation_data=(x_test, y_test))
```

After training our model, in scikit-learn, we can use 'score' function to show the accuracy on both training and test data.

```
print('\n')
print("Training accuracy: %f" % mlp.score(X_train, y_train))
print("Test accuracy: %f" % mlp.score(X_test, y_test))
```

```
Iteration 1, loss = 0.60665357
Iteration 2, loss = 0.41250209
Iteration 3, loss = 0.41051453
Iteration 4, loss = 0.42239682
Iteration 5, loss = 0.40242161
Iteration 6, loss = 0.41318707
Iteration 7, loss = 0.44401505
Iteration 8, loss = 0.47749627
Iteration 9, loss = 0.46657552
Iteration 10, loss = 0.46982848
Iteration 11, loss = 0.45269498
Iteration 12, loss = 0.45404194
Iteration 13, loss = 0.44569686
Iteration 14, loss = 0.45222837
Iteration 15, loss = 0.52326494
Iteration 16, loss = 0.48696374
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.


Training accuracy: 0.881167
Test accuracy: 0.878400
```

As can be seen from the image above, we have around 88.1% and 87.8% accuracy on train and test data respectively.

For keras, we can evaluate our model by calling 'evaluate' function.

```
score = model.evaluate(x_test, y_test, verbose=0)

print('\n')
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.11046630860159194
Test accuracy: 0.9837
```

As can be seen from the image above, we get the accuracy of around 98.3% on the test data. I suppose this is because I used more layers and neurons for keras experiment than for scikit-learn experiment.