04-HOMEWORK-MultilinearRegressionAndSupervisedLearning

November 18, 2021

1 Multilinear Regression and Supervised Learning

In this assignment, you will explore the regression and classification models that we discussed in class. While the class discussion was focused on theory, the focus here is on applying the methods to data.

We'll use a couple of new packages for loading and manipulating data: - `seaborn` is mostly for visualization of statistical analyses of data, but has some useful datasets that can be loaded from the package - `pandas` is a useful tool for working with spreadsheet-style data (technically, the data type used here is called a "dataframe")

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

1.1 Multilinear Regression

1.1.1 Loading Data

We'll load a dataset containing various statistics about car models, including their typical miles-per-gallon (mpg).

```
[2]: mpg_data = sns.load_dataset("mpg")
```

To get an idea for what the data looks like, we can look at the first few lines using the following command.

```
[3]: mpg_data.head()
```

```
[3]:     mpg  cylinders  displacement  horsepower  weight  acceleration  \
     0  18.0          8         307.0       130.0    3504          12.0
     1  15.0          8         350.0       165.0    3693          11.5
     2  18.0          8         318.0       150.0    3436          11.0
     3  16.0          8         304.0       150.0    3433          12.0
     4  17.0          8         302.0       140.0    3449          10.5

        model_year origin                       name
     0          70    usa  chevrolet chevelle malibu
     1          70    usa          buick skylark 320
```

```
2          70     usa         plymouth satellite
3          70     usa              amc rebel sst
4          70     usa                ford torino
```

We can see that there are 398 data samples with 9 pieces of information per sample.

[4]:
```
mpg_data.shape
```

[4]:
```
(398, 9)
```

However, if we look through the data, we see that some entries in the ‘horsepower’ column are not numbers.

[5]:
```
np.array(mpg_data['horsepower'])
```

[5]:
```
array([130., 165., 150., 150., 140., 198., 220., 215., 225., 190., 170.,
       160., 150., 225.,  95.,  95.,  97.,  85.,  88.,  46.,  87.,  90.,
        95., 113.,  90., 215., 200., 210., 193.,  88.,  90.,  95.,  nan,
       100., 105., 100.,  88., 100., 165., 175., 153., 150., 180., 170.,
       175., 110.,  72., 100.,  88.,  86.,  90.,  70.,  76.,  65.,  69.,
        60.,  70.,  95.,  80.,  54.,  90.,  86., 165., 175., 150., 153.,
       150., 208., 155., 160., 190.,  97., 150., 130., 140., 150., 112.,
        76.,  87.,  69.,  86.,  92.,  97.,  80.,  88., 175., 150., 145.,
       137., 150., 198., 150., 158., 150., 215., 225., 175., 105., 100.,
       100.,  88.,  95.,  46., 150., 167., 170., 180., 100.,  88.,  72.,
        94.,  90.,  85., 107.,  90., 145., 230.,  49.,  75.,  91., 112.,
       150., 110., 122., 180.,  95.,  nan, 100., 100.,  67.,  80.,  65.,
        75., 100., 110., 105., 140., 150., 150., 140., 150.,  83.,  67.,
        78.,  52.,  61.,  75.,  75.,  75.,  97.,  93.,  67.,  95., 105.,
        72.,  72., 170., 145., 150., 148., 110., 105., 110.,  95., 110.,
       110., 129.,  75.,  83., 100.,  78.,  96.,  71.,  97.,  97.,  70.,
        90.,  95.,  88.,  98., 115.,  53.,  86.,  81.,  92.,  79.,  83.,
       140., 150., 120., 152., 100., 105.,  81.,  90.,  52.,  60.,  70.,
        53., 100.,  78., 110.,  95.,  71.,  70.,  75.,  72., 102., 150.,
        88., 108., 120., 180., 145., 130., 150.,  68.,  80.,  58.,  96.,
        70., 145., 110., 145., 130., 110., 105., 100.,  98., 180., 170.,
       190., 149.,  78.,  88.,  75.,  89.,  63.,  83.,  67.,  78.,  97.,
       110., 110.,  48.,  66.,  52.,  70.,  60., 110., 140., 139., 105.,
        95.,  85.,  88., 100.,  90., 105.,  85., 110., 120., 145., 165.,
       139., 140.,  68.,  95.,  97.,  75.,  95., 105.,  85.,  97., 103.,
       125., 115., 133.,  71.,  68., 115.,  85.,  88.,  90., 110., 130.,
       129., 138., 135., 155., 142., 125., 150.,  71.,  65.,  80.,  80.,
        77., 125.,  71.,  90.,  70.,  70.,  65.,  69.,  90., 115., 115.,
        90.,  76.,  60.,  70.,  65.,  90.,  88.,  90.,  90.,  78.,  90.,
        75.,  92.,  75.,  65., 105.,  65.,  48.,  48.,  67.,  67.,  67.,
        nan,  67.,  62., 132., 100.,  88.,  nan,  72.,  84.,  84.,  92.,
       110.,  84.,  58.,  64.,  60.,  67.,  65.,  62.,  68.,  63.,  65.,
        65.,  74.,  nan,  75.,  75., 100.,  74.,  80.,  76., 116., 120.,
       110., 105.,  88.,  85.,  88.,  88.,  88.,  85.,  84.,  90.,  92.,
        nan,  74.,  68.,  68.,  63.,  70.,  88.,  75.,  70.,  67.,  67.,
        67., 110.,  85.,  92., 112.,  96.,  84.,  90.,  86.,  52.,  84.,
        79.,  82.])
```

We'll remove any rows from our data where `nan` appears.

```
[6]: mpg_data = mpg_data.dropna()
```

```
[7]: mpg_data.shape
```

```
[7]: (392, 9)
```

Now let's fit this into the regression framework that we described in class.

We will train our model to predict mpg from the remaining numerical variables (cylinders, displacement, horsepower, weight, acceleration, model_year). This means that our data matrix should be of the form

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_{398}^T \end{bmatrix} \in \mathbb{R}^{398\times 6},$$

where each $\vec{x}_i$ is a vector of 6 numerical measurements for car model i. The vector we are trying to predict is of the form

$$\vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{398} \end{bmatrix} \in \mathbb{R}^{398},$$

where each y_i is the mpg for care model i.

Let's set these up as `numpy` arrays:

```
[14]: X = np.
       ↪array(mpg_data[['cylinders','displacement','horsepower','weight','acceleration','model_year
      y = np.array(mpg_data['mpg'])
```

1.1.2 Building a Multilinear Regression Model

Recall the general setup for multilinear regression: give a data matrix $X \in \mathbb{R}^{N\times d}$ with N samples and d features and a vector of observations $\vec{y} \in \mathbb{R}^N$, our goal is to find parameters $\beta_0, \beta_1, \ldots, \beta_d$ such that the loss function

$$L(\beta_0, \ldots, \beta_d) = \sum_{i=1}^{N} \left(y_i - (\beta_0 + \beta_1(\vec{x}_i)_1 + \cdots + \beta_d(\vec{x}_i)_d)\right)^2.$$

In this formula, y_i is the ith entry of $\vec{y}$ and $\vec{x}_i$ is the ith data sample (so $\vec{x}_i^T$ is the ith row of X).

We will now build our model, which means finding the best parameters. There are a couple of sublties that we didn't address in class, which we will handle now.

1.1.3 Adding Bias

When we derived our formula for optimal parameters in multilinear regression in class, we didn't include a β_0 term (this is sometimes referred to in machine learning as the *bias*). To fix this, all we need to do is add a column of all ones to our data matrix X. After we do this, all of the other analysis that we did in class will still go through.

Problem 1. Fix the data matrix `X` defined above by adding a column of ones. Do this so that the *first* column contains zeros. That is, the current $X \in \mathbb{R}^{398\times 6}$ should be replaced by a new $X \in \mathbb{R}^{398\times 7}$ whose first column contains all ones and whose remaining columns are copied over from the original data matrix.

Note: Code below will assume that your new data matrix is still called `X`.

[33]:
```
## Your code goes here
X = np.concatenate((np.ones((392, 1)),X), axis=1)
```

1.1.4 Train-Test Split

Typically when one trains a model, it is trained on a subset of the data. The resulting model can then be tested on the remaining data that it did not see during the training phase. The following code picks out a percentage of rows from X randomly, together with corresponding entries of $\vec{y}$, as the 'training set' and leaves the rest out as a 'testing set'.

[76]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

1.1.5 Training the Model

Now recall that we found an explicit formula for the optimal parameters in the multilinear regression problem. Let $\vec{\beta} = [\beta_0\beta_1\cdots\beta_d]^T$ be the optimal parameter vector. It is given by the formula

$$\vec{\beta} = \left(X^T X\right)^{-1} X^T \vec{y}.$$

Problem 2. Find the optimal parameters for the training set using the formula above. Call your optimal parameter vector `betas`. Print out your answer.

[77]:
```
## Your code goes here.
betas = np.linalg.inv(X_train.T@X_train)@X_train.T@y_train
print(betas)
```

```
[-1.60611943e+01 -5.42058921e-01  1.62545504e-02  9.79855738e-03
 -7.72828049e-03  2.21488364e-01  7.64821335e-01]
```

We can test the performance of our model by making predictions on the testing data, which was not incorporated into our training step.

Problem 3. Create a vector `y_pred` containing the predicted `mpg` values of the data from your testing set.

Hint: This can be done by multiplying a certain matrix with a certain vector.

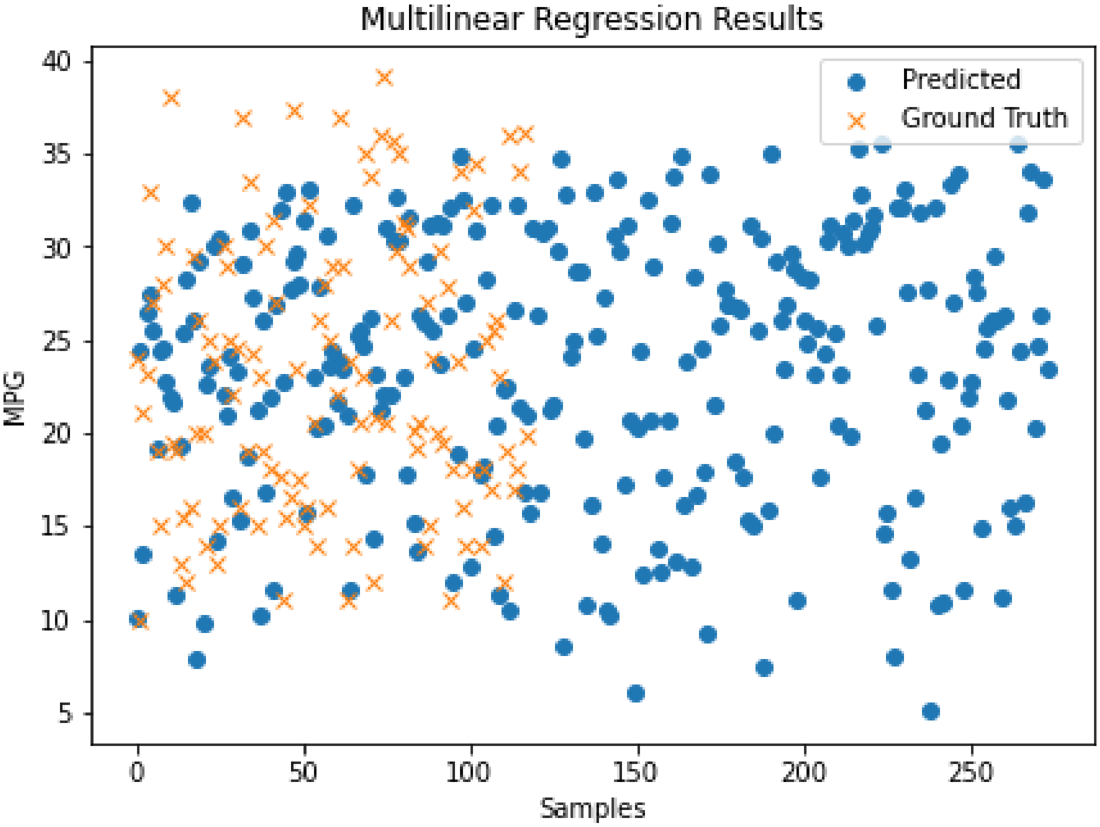
[78]:

```
## Your code goes here.
y_pred = X_train@betas
```

The code below plots the predicted `mpg` values on the test set, versus the true values from `y_test`. The results look reasonable, but to get a real feel for the quality of our results, we would want to test against other models.

[79]:

```
plt.figure(figsize = (7,5))
plt.plot(y_pred,'o', label = 'Predicted')
plt.plot(y_test,'x', label = 'Ground Truth')
plt.title('Multilinear Regression Results')
plt.xlabel("Samples")
plt.ylabel("MPG")
plt.legend()
plt.show()
```



1.2 Supervised Learning for Classification

The multilinear regression model we created above is designed for predicting a continuous observation. Many machine learning models are designed to predict labels for labeled data (e.g., images). We will now explore some of these methods.

1.2.1 Loading the Data

For this part of the notebook, we'll use the (small, built-in `sklearn` version of the) MNIST dataset of handwritten digits.

[67]:

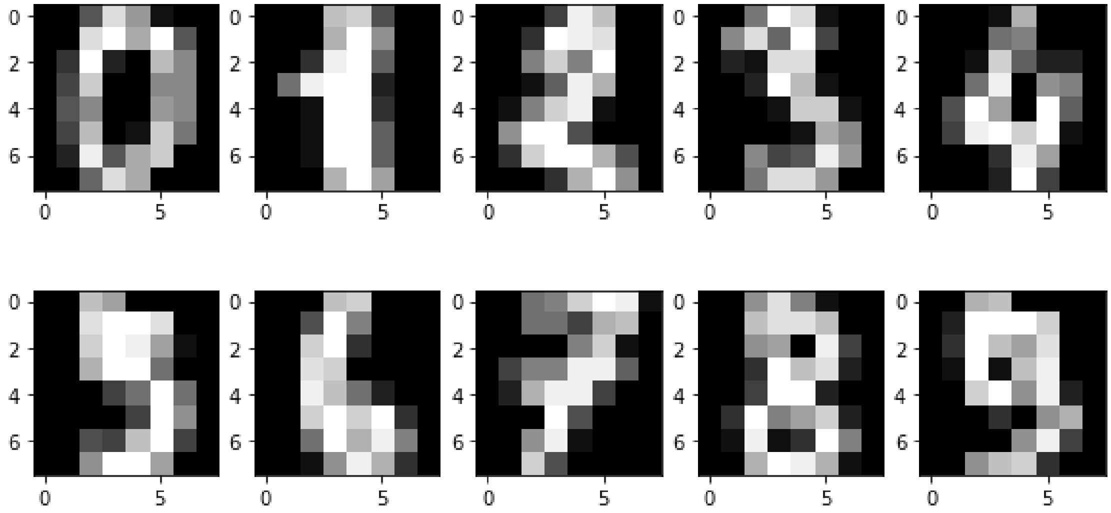
```
from sklearn import datasets
digits = datasets.load_digits()
```

Recall that the data consists of many handwritten digits and their correct labels.

[68]:

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

for j in range(10):
    fig.add_subplot(2,5,j+1)
    plt.imshow(digits.images[j], cmap='gray')
```



We can treat each image as a vector in $\mathbb{R}^{8\times 8} \approx \mathbb{R}^{64}$. Conveniently, `digits.data` reshapes each image into a 64 dimensional vector. Let's create a train/test split of this vectorized data.

[69]:

```
MNIST = digits.data
MNISTlabels = digits.target

MNIST_train, MNIST_test, MNISTlabels_train, MNISTlabels_test =␣
 ↪train_test_split(MNIST, MNISTlabels)
```

As we discussed in class, we can use Logistic Regression for classification. The version that we discussed in class was only for classifying datasets with two labels ('cats' versus 'dogs', say), but this can be generalized to handle datasets with an arbitrary number of labels. In this case, we have 10 labels (one for each digit).

The code below trains the logistic regression model on the training data (i.e., finds the best parameters for the model). This is standard `sklearn` syntax. Note that this training process uses a version of gradient descent to find optimal values, rather than an explicit formula as in the multilinear regression example.

[70]:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', multi_class = 'multinomial',␣
 ↪max_iter=10000)
model.fit(MNIST_train, MNISTlabels_train)
```

[70]:

```
LogisticRegression(max_iter=10000, multi_class='multinomial')
```

Logistic regression with two labels works by finding a good hyperplane to split the two classes of data vectors (for precise definitions, refer back to the lecture notes). For multiclass logistic regression, we get a hyperplane for each class label. This can be seen in the shape of the coefficients for the model:

[71]:

```
model.coef_.shape
```

[71]:

```
(10, 64)
```

Now we can predict a label for each digit in our testing set. This is done using the built-in `predict` function for the model.

[73]:

```
MNIST_predicted = model.predict(MNIST_test)
```

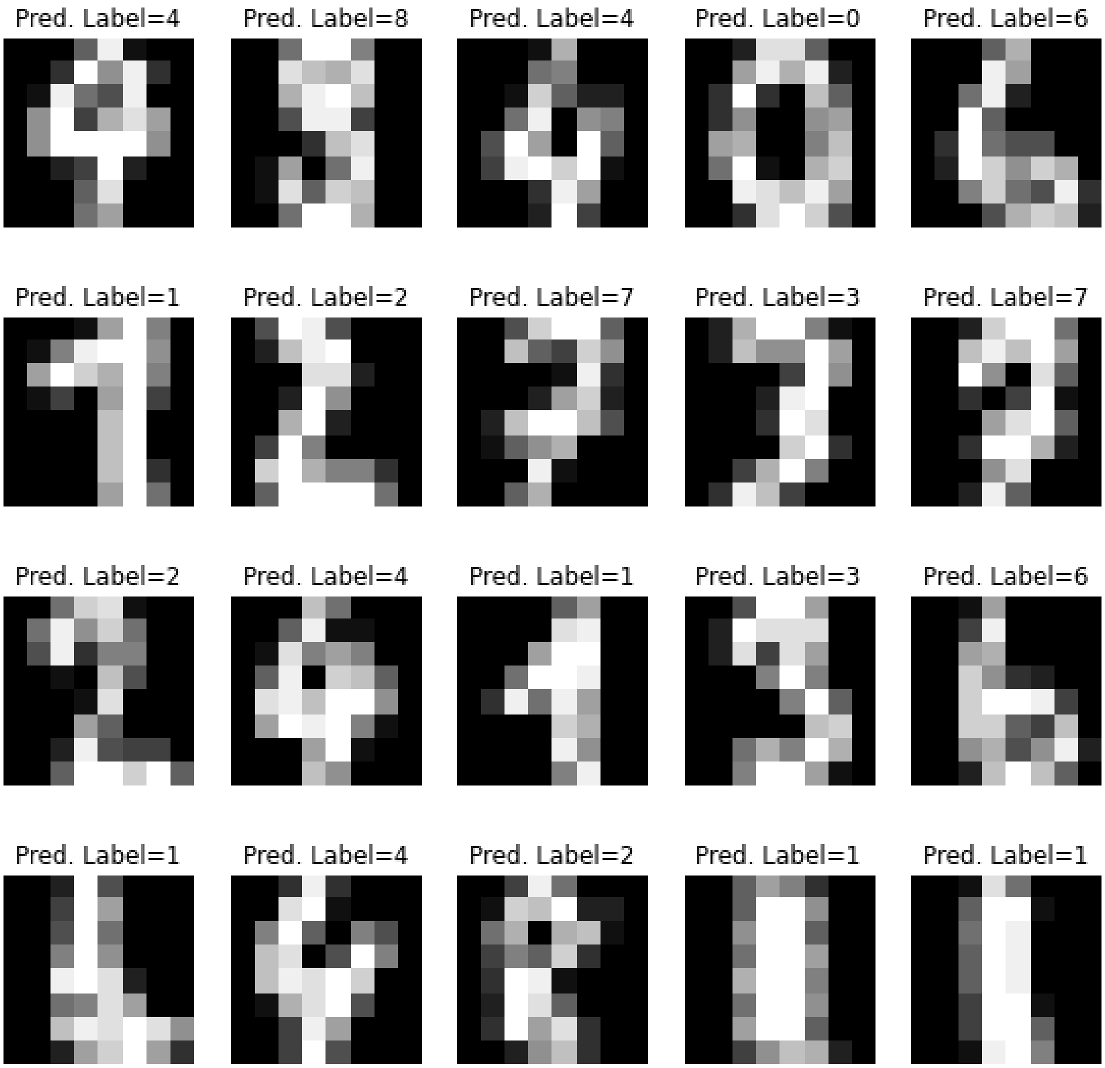
Let's take a look at some of the samples from our testing set, along with their predicted labels.

[74]:

```
fig = plt.figure(figsize=(10,10))

for j in range(20):
    fig.add_subplot(4,5,j+1)
    plt.imshow(MNIST_test[j].reshape(8,8), cmap='gray')
    plt.axis('off')
    plt.title('Pred. Label='+str(MNIST_predicted[j]))

plt.show()
```



These results look pretty good. We can compute the overall prediction accuracy using the `score` function.

```
[75]: print('Prediction Accuracy:',np.round(model.
      ↪score(MNIST_test,MNISTlabels_test),2))
```

```
Prediction Accuracy: 0.95
```

Logistic regression performs extremely well! It will probably be hard to beat this, but let's practice with some other models:

Problem 4. Repeat the above classification process using a different classification model. You can take your pick from Multilayer Perceptron (a simple neural network), Decision Tree, Support Vector Machine, or any other model that you like.

Remark: If you haven't done this before, you'll need to dig into the documentation a little bit to see how these work. The syntax is mostly similar to the logistic regression model. The mathematical details of several of these models will be described in final project presentations!

[ ]:

```
## Your code goes here.
```

[152]:

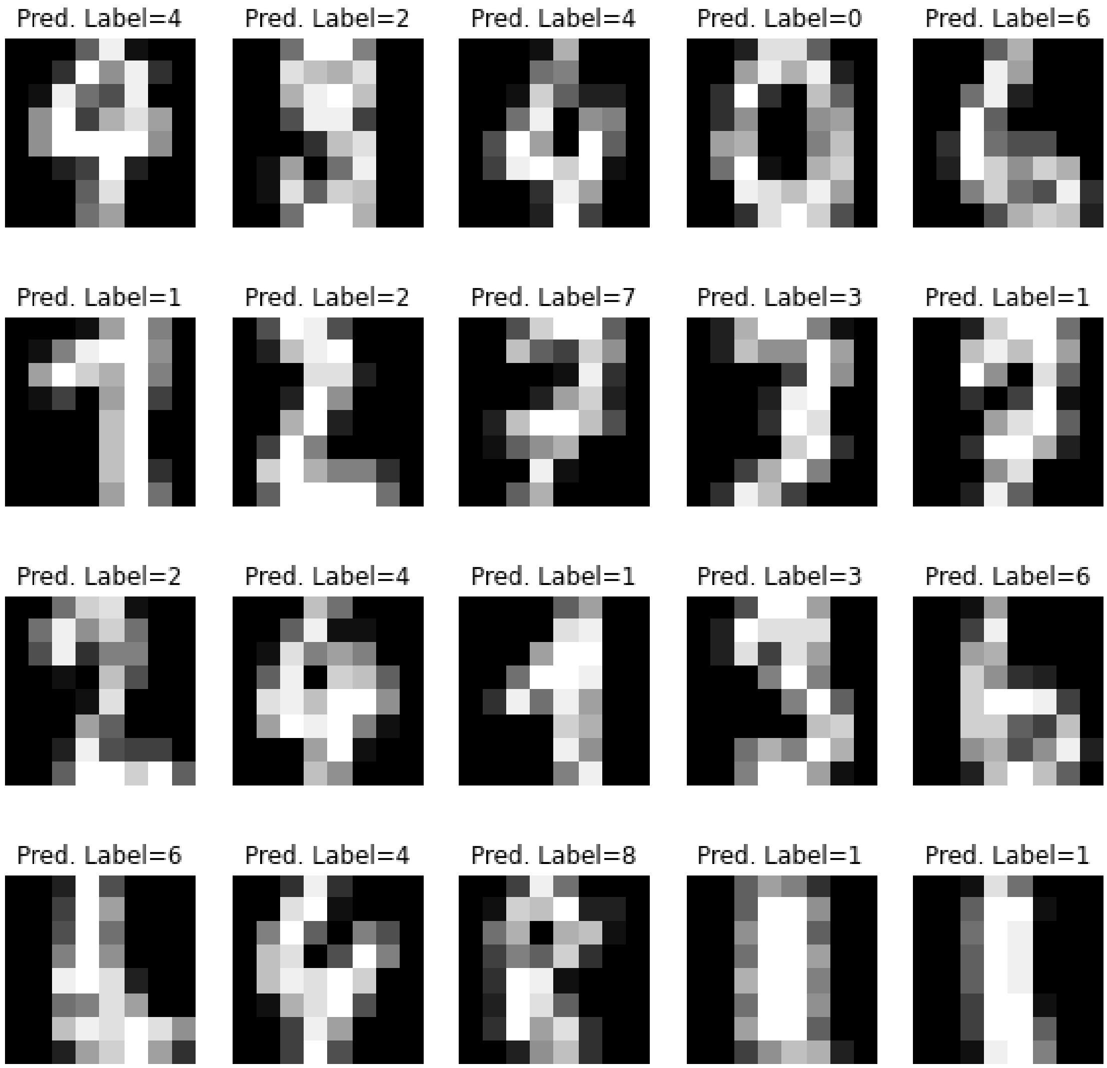
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(15,), random_state=1, max_iter=10000)
mlp.fit(MNIST_train, MNISTlabels_train)
mlp_MNIST_predict = mlp.predict(MNIST_test)
```

[153]:

```
fig = plt.figure(figsize=(10,10))

for j in range(20):
    fig.add_subplot(4,5,j+1)
    plt.imshow(MNIST_test[j].reshape(8,8), cmap='gray')
    plt.axis('off')
    plt.title('Pred. Label='+str(mlp_MNIST_predict[j]))

plt.show()
```


Pred. Label=4
Pred. Label=2
Pred. Label=4
Pred. Label=0
Pred. Label=6
Pred. Label=1
Pred. Label=2
Pred. Label=7
Pred. Label=3
Pred. Label=1
Pred. Label=2
Pred. Label=4
Pred. Label=1
Pred. Label=3
Pred. Label=6
Pred. Label=6
Pred. Label=4
Pred. Label=8
Pred. Label=1
Pred. Label=1