Handwritten Digit Recognition on MNIST dataset

By :- 1.) Rishabh Tenguria(B190062EC)
2.) Saurav Kumar (B190065EC)
3.) Meenu Yadav(B190057EC)

What is MNIST?

- Set of 70,000 small images of digits handwritten by high school students and employees of the US causes Bureau.
- All images are labeled with the respective digit they represent.
- MNIST is the hello world of machine learning. Every time a data scientist or machine learning engineer makes a new algorithm for classification, they would always first check its performance on the MNIST dataset.
- There are 70,000 images and each image has 28*28 = 784 features.
- Each image is 28*28 pixels and each feature simply represents one-pixel intensity from 0 to 255. If the intensity is 0, it means that the pixel is white and if it is 255, it means it is black.

Continue..

- Refer to the image below displaying handwritten digits in the MNIST dataset taken from Wikipedia:

- 1. First of all, import all the libraries required. We will import the *fetch_openml* from the sklearn.datasets library.
- 2. Create a variable *mnist*, and store in it the ***mnsit_784* dataset** from the ***featch_openml*** And wecan further print and see the contents of this *mnist* dataset. we can see its keys, its data, its corresponding labels, and more.

```python
# fetching dataset
from sklearn.datasets import fetch_openml
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

mnist = fetch_openml('mnist_784')
```

- 3. Create array variables *x and y.* Store in them the data and the targets respectively of the *mnist* As we discussed above, we have 784 (28x28) pixels of features, and these are now stored in *x.* Variable *y* has the corresponding digit that the picture resembled by *x* contains.
- 4. we can try to see that picture *x* using *matplotlib* Since the pixels here are stacked together in a 1D array *x* for memory issues, you'll have to reshape it back to 26x26. Create a variable *some_digit* and fill it with any random digit array from the dataset. Reshape it and store it in another variable *some_digit_image.*

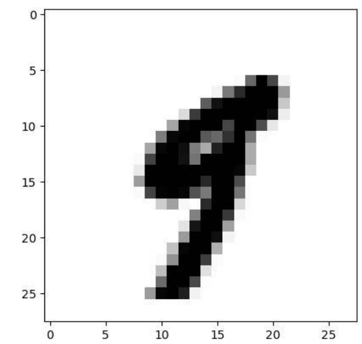
- 5. we can now simply use the command below to get that image plotted. We have used the *imshow* attribute of pyplot and have fed the *some_digit_image* 2D array of pixel data into it.

```
x, y = mnist['data'], mnist['target']

some_digit = x.to_numpy()[36001]
some_digit_image = some_digit.reshape(28, 28)  # let's reshape to plot it

plt.imshow(some_digit_image, cmap=matplotlib.cm.binary,
           interpolation='nearest')
plt.axis("off")
plt.show()
```

6. And since the data, we stored corresponded to the 36000th element, the image we were shown was:



- 7. You can turn the axes off, using the axis property of pyplot. And now we can see what digits these images resemble. You must have already observed that by now, the first image was a 9, and the second image was a 2. Verify it by printing the labels stored in y.

- 8. And for our convenience, MNIST dataset is already split into training and testing data. The first 60000 are training data, and the last 10000 are testing data. Create two array variables *x_train and x_test* and store in them, the training and the testing data respectively. Similarly, create another two array variables *y_train and y_test* and store in them, the training and the testing labels/targets respectively. And once you split your data, you must consider shuffling the dataset.

- 9. Now, since there are 10 different labels from 0 to 9, and we want to do binary classification, we would replace our target from 0 to 9 to true and false, if the target is a 2 or not. Create two array variables *y_train_2 and y_test_2* and store in them the true false value of *y_train and y_test* if the label is a 2 or not.

```
x_train, x_test = x[:60000], x[6000:70000]
y_train, y_test = y[:60000], y[6000:70000]

shuffle_index = np.random.permutation(60000)
x_train, y_train = x_train.[shuffle_index], y_train.[shuffle_index]

# Creating a 2-detector
y_train = y_train.astype(np.int8)
y_test = y_test.astype(np.int8)
y_train_2 = (y_train == '2')
y_test_2 = (y_test == '2')
```

- Now, import another package from sklearn.linear_model called LogisticRegression and load it into a classifier variable *clf*. This creates our classifier. Use the *fit* attribute of the classifier model to feed our features and labels into it and the *predict* attribute to predict the labels based on some features. And after our classifier gets trained, we'll test if it predicts true for the image 36001.

```
# Train a logistic regression classifier
clf = LogisticRegression(tol=0.1)
clf.fit(x_train, y_train_2)
example = clf.predict([some_digit])
print(example)
```

- And the output we received was [True].
- Hence the feature falls into the category of digit 2. So, this is how we build and train a logistic regression on MNIST dataset, and we did show you how we predict the labels using it.

- **Cross Validating:**
- 11. As we studied earlier, cross-validation increases the efficiency of the model. We would cross-validate our model, and retrieve the accuracy of the model. Don't forget to import *cross_val_score* from sklearn.model_selection. Create a variable *a* and store in it the accuracy as measured by our cross validator, when we passed our model, and the training data. Check its mean for the overall accuracy of the model.

```
# Cross Validation
a = cross_val_score(clf, x_train, y_train_2, cv=3, scoring="accuracy")
print(a.mean())
```

- Our output was 0.9781333333333334 which is quite high. Having an accuracy of 97.8% is indeed a great deal. But I would like to give you all your homework before we end this lecture. Create a classifier that will classify a digit always as not 2. Now, there is a spoiler. Since more than 90% of the digits here are not 2, and even if you classify all the digits as not 2, you would get an accuracy of 90% at least. This was a catch. Is accuracy always a great metric to define a good classifier? NO.

Thank You