

Guided Project Report

Handwritten Digit Recognition

Name: Shruti Verma

Course: AI and ML

(Batch 4)

Duration: 10 months

Problem Statement: Implement the classifier by squeezing the image into a vector and then using a MLP and SVM

Prerequisites

What things you need to install the software and how to install them:

Python 3.8 or higher versions This setup requires that your machine has latest version of python. The following url <https://www.python.org/downloads/> can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: <https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/>. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic.

Second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url <https://www.anaconda.com/download/> You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.8 then run below commands in command prompt/terminal to install these packages pip install -U scikit-learn pip install numpy pip install scipy if you have chosen to install anaconda then run below commands in anaconda prompt to install these packages conda install -c scikit-learn conda install -c anaconda numpy conda install -c anaconda scipy . Install the keras and sklearn.svm import SVC .

Video Link

<https://drive.google.com/drive/folders/1nKjHKic6ZATLwmXNwcyb1Guxb1LNOrrO>

Dataset used

Using the MNIST dataset, training set of 60,000 examples and test set of 10,000 examples.

Method used for detection

- Reading the dataset, squeezing the image
- Converting to vector
- Applying model

Importing the libraries and capturing images:

Importing the necessary libraries and reading the MNIST train and test data

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [1]:  
1 # importing the necessary libraries  
2 import pandas as pd  
3 import numpy as np  
4  
5 np.random.seed(1212)  
6  
7 import keras  
8 from keras.models import Model  
9 from keras.layers import *  
10 from keras import optimizers  
  
In [2]:  
1 df_train = pd.read_csv('train.csv')  
2 df_test = pd.read_csv('test.csv')  
  
In [3]:  
1 df_train.head()  
  
Out[3]:  
label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 ... pixel774 pixel775 pixel776 pixel777 pixel778 pixel779 pixel780 pixel781 pixel782  
0 1 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
1 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
2 1 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
3 4 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
4 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
  
5 rows × 785 columns  
  
In [4]:  
1 # Splitting the training and validation dataset  
2  
3 df_features = df_train.iloc[:, 1:785]  
4 df_label = df_train.iloc[:, 0]  
5  
6 X_test = df_test.iloc[:, 0:784]  
7  
8 print(X_test.shape)  
(28000, 784)
```

Splitting the data into train and test and checking the labels and features

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook has tabs for "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A "Trusted" button and "Python 3" are also visible.

```

In [4]: 1 # Splitting the training and validation dataset
2
3 df_features = df_train.iloc[:, 1:785]
4 df_label = df_train.iloc[:, 0]
5
6 X_test = df_test.iloc[:, 0:784]
7
8 print(X_test.shape)

(28000, 784)

```

```

In [5]: 1 df_label

Out[5]: 0      1
1      0
2      1
3      4
4      0
.. .
41995  0
41996  1
41997  7
41998  6
41999  9
Name: label, Length: 42000, dtype: int64

```

```

In [6]: 1 from sklearn.model_selection import train_test_split
2 X_train, X_cv, y_train, y_cv = train_test_split(df_features, df_label,
3                                               test_size = 0.2,
4                                               random_state = 1212)
5
6

```

Data processing and using one hot encoder for labels

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook has tabs for "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A "Trusted" button and "Python 3" are also visible.

```

In [11]: 1 X_train = X_train.to_numpy().reshape(33600, 784)

In [12]: 1 X_train.reshape(33600, 784)
2 X_train.shape

Out[12]: (33600, 784)

In [13]: 1 X_cv = X_cv.to_numpy().reshape(8400, 784)
2 X_cv.shape

Out[13]: (8400, 784)

In [15]: 1 X_test = X_test.to_numpy().reshape(28000, 784)
2 X_test.shape

Out[15]: (28000, 784)

In [16]: 1 #Data cleaning and normalize
2 print((min(X_train[1]), max(X_train[1])))

(0, 255)

```

```

In [17]: 1 # Feature Normalization
2 X_train = X_train.astype('float32'); X_cv= X_cv.astype('float32'); X_test = X_test.astype('float32')
3 X_train /= 255; X_cv /= 255; X_test /= 255
4
5 # Convert labels to One Hot Encoded
6 num_digits = 10
7 y_train = tf.keras.utils.to_categorical(y_train, num_digits)
8 y_cv = tf.keras.utils.to_categorical(y_cv, num_digits)

```

```

In [18]: 1 # Printing 2 examples of labels after conversion
2 print(y_train[0]) # 2
3 print(y_train[3]) # 7

[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]

```

Model Fitting

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook contains several code cells:

- In [18]:

```
1 # Printing 2 examples of labels after conversion
2 print(y_train[0]) # 2
3 print(y_train[3]) # 7
```

Output:

```
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
```
- In [19]:

```
1 # Model Fitting
2 # Input Parameters
3 n_input = 784 # number of features
4 n_hidden_1 = 300
5 n_hidden_2 = 100
6 n_hidden_3 = 100
7 n_hidden_4 = 200
8 num_digits = 10
```
- In [20]:

```
1 Inp = Input(shape=(784,))
2 x = Dense(n_hidden_1, activation='relu', name = "Hidden_Layer_1")(Inp)
3 x = Dense(n_hidden_2, activation='relu', name = "Hidden_Layer_2")(x)
4 x = Dense(n_hidden_3, activation='relu', name = "Hidden_Layer_3")(x)
5 x = Dense(n_hidden_4, activation='relu', name = "Hidden_Layer_4")(x)
6 output = Dense(num_digits, activation='softmax', name = "Output_Layer")(x)
7
8
```
- In [21]:

```
1 # Our model would have '6' layers - input layer, 4 hidden layer and 1 output layer
2 model = Model(Inp, output)
3 model.summary() # We have 297,910 parameters to estimate
```

Output:

Model: "model"		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
Hidden_Layer_1 (Dense)	(None, 300)	235500
Hidden_Layer_2 (Dense)	(None, 100)	30100
Hidden_Layer_3 (Dense)	(None, 100)	10100

Checking the loss and accuracy of the model

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook contains several code cells:

- In [22]:

```
1 # Insert Hyperparameters
2 learning_rate = 0.1
3 training_epochs = 20
4 batch_size = 100
5 sgd = tf.keras.optimizers.SGD(lr=learning_rate)
6 sgd
```

Output:

```
/Users/ashokdayal/opt/anaconda3/envs/shru/lib/python3.8/site-packages/keras/optimizer_v2/optimizer_v2.py:355: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
warnings.warn(
Out[22]: <keras.optimizer_v2.gradient_descent.SGD at 0x14e78f820>
```
- In [23]:

```
1 # We rely on the plain vanilla Stochastic Gradient Descent as our optimizing methodology
2 model.compile(loss='categorical_crossentropy',
3                 optimizer='sgd',
4                 metrics=['accuracy'])
```
- In [24]:

```
1 X_train.shape
Out[24]: (33600, 784)
```
- In [25]:

```
1 del.fit(X_train, y_train,batch_size = batch_size,epochs = training_epochs,verbose = 2,validation_data=(X_cv, y_cv))
2
336/336 - 1s - loss: 0.1911 - accuracy: 0.9440 - val_loss: 0.2027 - val_accuracy: 0.9388
Epoch 12/20
336/336 - 1s - loss: 0.1804 - accuracy: 0.9474 - val_loss: 0.1967 - val_accuracy: 0.9427
Epoch 13/20
336/336 - 1s - loss: 0.1690 - accuracy: 0.9501 - val_loss: 0.1909 - val_accuracy: 0.9440
Epoch 14/20
336/336 - 1s - loss: 0.1591 - accuracy: 0.9529 - val_loss: 0.1790 - val_accuracy: 0.9467
Epoch 15/20
336/336 - 1s - loss: 0.1511 - accuracy: 0.9547 - val_loss: 0.1701 - val_accuracy: 0.9477
Epoch 16/20
336/336 - 1s - loss: 0.1423 - accuracy: 0.9578 - val_loss: 0.1693 - val_accuracy: 0.9502
Epoch 17/20
336/336 - 1s - loss: 0.1350 - accuracy: 0.9602 - val_loss: 0.1657 - val_accuracy: 0.9500
Epoch 18/20
336/336 - 1s - loss: 0.1283 - accuracy: 0.9618 - val_loss: 0.1549 - val_accuracy: 0.9536
Epoch 19/20
336/336 - 1s - loss: 0.1213 - accuracy: 0.9643 - val_loss: 0.1504 - val_accuracy: 0.9554
2
```

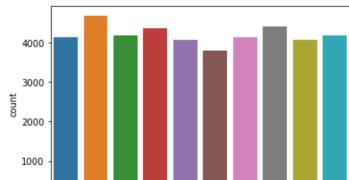
Using SVC to check the performance difference

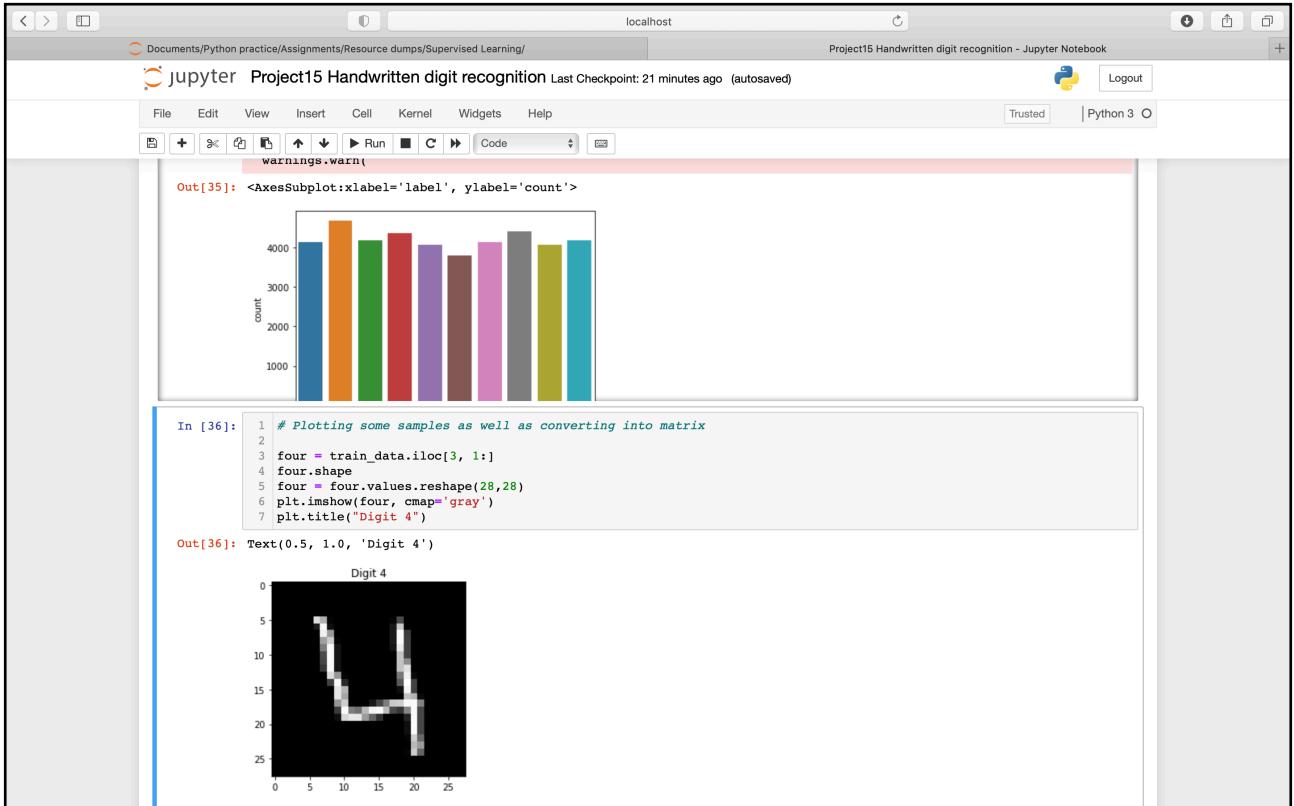
The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition - Jupyter Notebook". The code in cell In [26] imports various Python libraries including numpy, pandas, and scikit-learn. Cells In [27] and In [28] read CSV files for training and testing data. Cell In [29] prints the shape of the training data, which is (42000, 785). Cell In [30] prints the first five columns of the training data, showing a DataFrame with columns labeled 'label' and 'pixel0' through 'pixel781'. Cell In [31] prints the first five columns of the testing data.

```
In [26]:  
1 import numpy as np  
2 import pandas as pd  
3 from sklearn.model_selection import train_test_split  
4 from sklearn.svm import SVC  
5 from sklearn.metrics import confusion_matrix  
6 from sklearn.model_selection import validation_curve  
7 from sklearn.model_selection import KFold  
8 from sklearn.model_selection import cross_val_score  
9 from sklearn.model_selection import GridSearchCV  
10 import matplotlib.pyplot as plt  
11 import seaborn as sns  
  
In [27]:  
1 train_data = pd.read_csv("train.csv") #reading the csv files using pandas  
2 test_data = pd.read_csv("test.csv")  
  
In [28]:  
1 train_data.shape # print the dimension or shape of train data  
Out[28]: (42000, 785)  
  
In [29]:  
1 test_data.shape # print the dimension or shape of test data  
Out[29]: (28000, 784)  
  
In [30]:  
1 train_data.head() # printing first five columns of train_data  
  
Out[30]:  
label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 ... pixel774 pixel775 pixel776 pixel777 pixel778 pixel779 pixel780 pixel781 pixel782  
0 1 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
1 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
2 1 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
3 4 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
4 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0  
5 rows × 785 columns  
  
In [31]:  
1 test_data.head() # printing first five columns of test_data
```

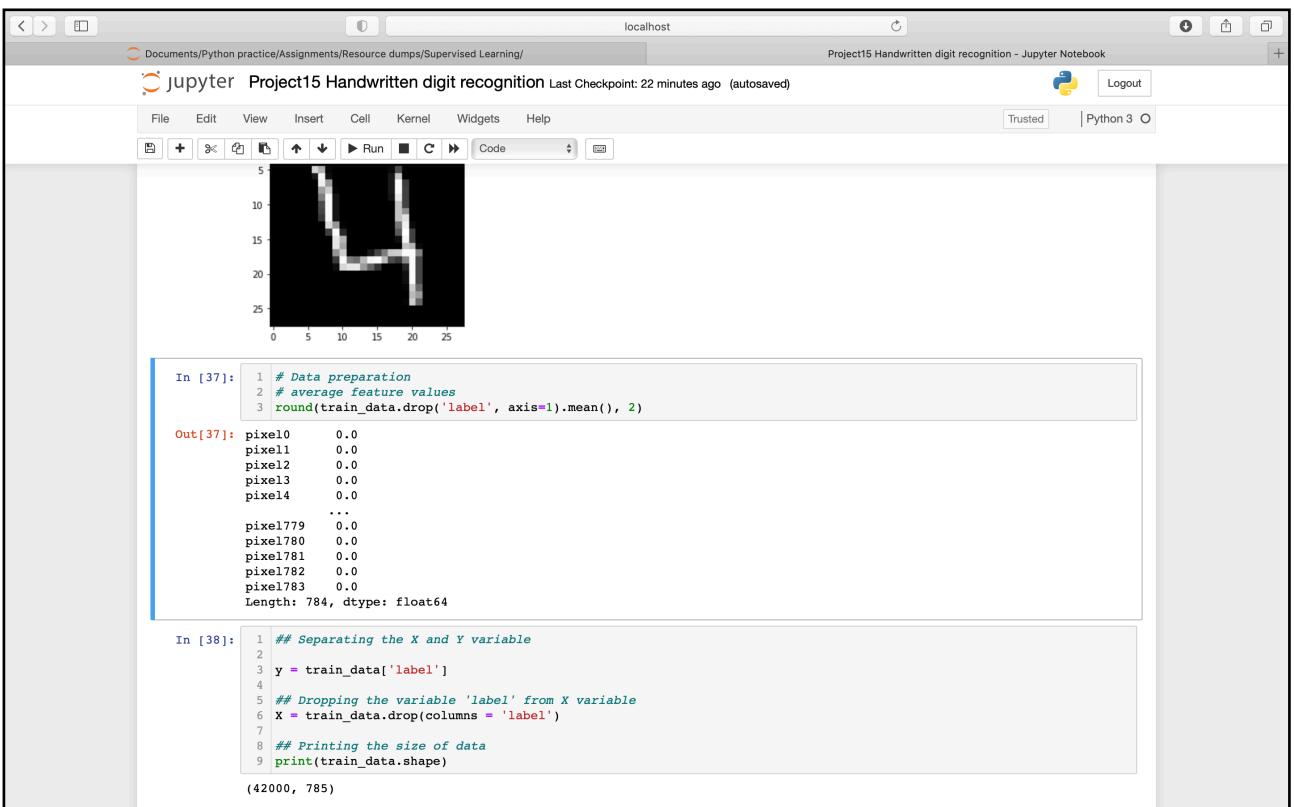
Visual for number count and number of classes, covering to matrix

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition - Jupyter Notebook". Cell In [34] sorts the unique labels in the training data. Cell In [35] visualizes the number of classes and counts using a bar chart. The chart has 'label' on the x-axis and 'count' on the y-axis, ranging from 0 to 4000. The bars are colored in various shades of orange, green, red, purple, brown, grey, yellow, and blue.

```
In [34]:  
1 order = list(np.sort(train_data['label'].unique()))  
2 print(order)  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
In [35]:  
1 ## Visualizing the number of class and counts in the datasets  
2  
3  
4 sns.countplot(train_data["label"])  
  
/Users/ashokdayal/opt/anaconda3/envs/shru/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.  
warnings.warn(  
Out[35]: <AxesSubplot:xlabel='label', ylabel='count'>  
  

```



Data Preparation



Model Fitting

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook has several cells:

- In [38]:

```
6 X = train_data.drop(columns = 'label')
7 ## Printing the size of data
8 print(train_data.shape)
(42000, 785)
```
- In [39]:

```
1 ## Normalization
2
3 X = X/255.0
4 test_data = test_data/255.0
5
6 print("X:", X.shape)
7 print("test_data:", test_data.shape)
```

X: (42000, 784)
test_data: (28000, 784)
- In [40]:

```
1 calling the features
2 m sklearn.preprocessing import scale
3 scaled = scale(X)
4
5 rain test split
6 rain, X_train, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.3, train_size = 0.2 ,random_state = 10)
```
- In [41]:

```
1 # linear model
2
3 model_linear = SVC(kernel='linear')
4 model_linear.fit(X_train, y_train)
5
6 # predict
7 y_pred = model_linear.predict(X_test)
```
- In [42]:

```
1 # confusion matrix and accuracy
2
3 from sklearn import metrics
4 from sklearn.metrics import confusion_matrix
5 # accuracy
6 print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
7 # cm
```

The screenshot shows a Jupyter Notebook interface with the title "Project15 Handwritten digit recognition". The notebook has several cells:

- In [43]:

```
1 # model with optimal hyperparameters
2
3 # model
4 model = SVC(C=10, gamma=0.001, kernel="rbf")
5
6 model.fit(X_train, y_train)
7 y_pred = model.predict(X_test)
8
9 # metrics
10 print("accuracy", metrics.accuracy_score(y_test, y_pred), "\n")
11 print(metrics.confusion_matrix(y_test, y_pred), "\n")
```

accuracy 0.9438888888888889

1163	0	4	1	1	2	8	6	3	0
0	1389	4	2	4	0	1	9	4	0
1	4	1184	14	5	1	9	30	7	51
0	3	15	1263	0	14	2	23	8	31
1	2	20	3	1149	0	10	10	2	21
2	8	3	30	4	1064	15	9	11	31
8	1	3	0	3	13	1167	23	1	0
4	9	10	8	12	0	0	1255	2	301
5	18	17	23	8	20	5	13	1098	101
5	3	2	27	21	1	1	51	3	1161
- In []: 1