Hochschule Bremen
City University of Applied Sciences

Assignment II

Image Processing and Pattern Recognition.
Winter Semester 2021
MSc E.E.

Lecturer: Yannik Steiniger

Date of Submission: 10/12/2021

Jose Luis Quinones Gonzalez - 5172885

Contents

1.Introduction.

The following document is centered to describe some methods, in the field of image processing, relating the HOG Features, Supervised Classifiers, and for evaluating classifier performance, Confusion Matrix, and Sklearn Metrics. The implemented software and libraries, packages are: Spyder v3.8 (Programming language: Python), “matplotlib”, “skimage”, “sklearn”.

This document demonstrates how to classify, evaluate, and process digits’ images mainly. The final purpose is to comprehend how these digits can be transformed into a data, training, and testing set so that these same can be later classified into a specific category, using supervised methods (in the field of machine learning) as Logistic Regression; afterwards, such results can be validated and evaluated using the sci-kit images modules, for instance, “metrics”, “clf” (estimator for classifying when implementing datasets, this servers for predicting category, or labeled class). In addition, the method of Histogram of Oriented Gradients is also put into practice. Nonetheless, this method is only to be studied and analyzed, this is not taken into consideration for the classification/ prediction of digits classes.

1.1Image Processing.

Image processing relates the fact of software implementation, or the use of coding for automatic processing, manipulation, analysis, and interpretation of images. [1]

1.2HOG Features.

Histogram of Gradients. The basic idea behind this method is the implementation of histogram gradients for describing the main features of an image. The process involves the conversion of a given image into small cells; every cell represents a set of pixels which is then transformed to a specific histogram of gradient directions. [2]

1.3Some Types of classifiers (Machine Learning).

A classifier in machine learning is a program or function which maps unlabeled or labeled values, variables, images, or instances; therefore, these can be sorted into classes/ categories. [3]

Main differences of Classifier Types [4]:

<table>
<tr><th>Type of Classifiers</th><th>Main Difference</th><th>Type of Task</th></tr>
<tr><td rowspan="2">Supervised Classifiers</td><td rowspan="2">Datasets are labeled</td><td>Classification</td></tr>
<tr><td>Regression</td></tr>
<tr><td rowspan="3">Unsupervised Classifiers</td><td rowspan="3">Datasets are not labeled</td><td>Clustering</td></tr>
<tr><td>Association</td></tr>
<tr><td>Dimensionality Reduction</td></tr>
</table>

In the current work, the use of a Supervised Classifier (type of task: Regression) is utilized for predicting the class of each digit within a specific dataset.

1.3.1Logistic Regression

The following Method can be expounded graphically as follows:

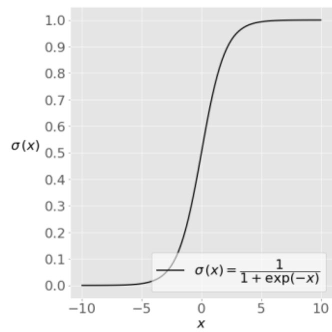


Figure 1. Logistic Regression Graph

The above mathematical function is vastly used when it comes to classify data due to its "S" form and borders; since the latter are close to 0 and 1, this serves as a perfect curve for classifying values or instances. [3]

Table 1. Activation Functions, Sigmoid and Relu

$$x_1 \rightarrow \quad x_2 \rightarrow \quad \cdots \quad x_n \rightarrow \qquad \xrightarrow[\text{Activation Function } \Phi]{\sum_{i=1}^{n}(w_i \cdot x_i)} \qquad \Phi\,(x_1, x_2 \cdots x_n) \rightarrow$$

Moreover, it is necessary to mention the activation function of sigmoid and Relu since the function logistic regression in Python is based upon this method. In brief, all input values go into a function, as to be more suitable to be processed, the results will be solely either 0 or 1. Such values will be then fitted into the borders of the "S" shape function, as the follow picture expounds:

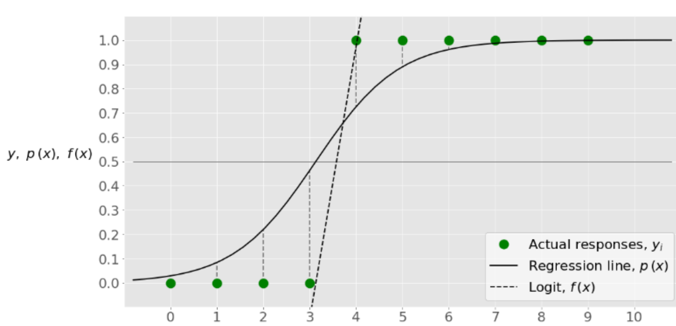


Figure 2. Example of Logistic Regression.

2.Tasks/Preparations.

1. Creating Training Data. (Implementing given MATLAB code).

Using the MATLAB code *Generate_the_training_data_A2.m,* a set of different digits, varying from 0 to 9 is created. The selected types of fonts are,

Table 2. Types of Fonts for Dataset

Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	Type 7	Type 8	Type 9	Type 10
Arial	Century	Courier	Times	Calibri	Calibri Light	Biome	Cambria	Cambria Math	Centaur

2. Creating Representative Data Set.

One set of 50 digits varying from 0 to 9 is created. To be more specific, this set has a pattern but different fonts. Every set is saved in a specific path for further utilization within the Spyder software. From this set another two more sets of 150 and 600 digits are created. These latter sets are made of random fonts and have also a random arrangement.

Table 3. Random Dataset 600 mixed fonts

Dataset	Values (Images, labels)
Case I. X_train (numpy array), y_train (Label) **50** X_test (numpy array), y_test (Label) **100**	[0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,6,6,6,6,6,7,7,7,7,7,8,8,8,8,8,9,9,9,9,9] [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9]
Case II. X (numpy array), y (Label) **150**	[0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0]
Case III. X (numpy array), y (Label) **600**	[0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,2,5,9,8,1,0,5,8,8,5,2,5,4,6,0,9,1,2,3,5,0,5,1,3,3,5,7,2,3,1,0,0,2,3,5,8,5,7,8,3,4,0,8,7,2,6,7,9,7,9,8,7,3,6,5,7,6,7,1,9,3,3,6,9,4,6,2,6,8,3,5,8,6,4,6,3,4,7,8,6,1,2,9,7,4,9,3,5,1,7,4,4,3,2,5,1,1,3,1,1,0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,2,5,9,8,1,0,5,8,8,5,2,5,4,6,0,9,1,2,3,5,0,5,1,3,3,5,7,2,3,1,0,0,2,3,5,8,5,7,8,3,4,0,8,7,2,6,7,9,7,9,8,7,3,6,5,7,6,7,1,9,3,3,6,9,4,6,2,6,8,3,5,8,6,4,6,3,4,7,8,6,1,2,9,7,4,9,3,5,1,7,4,4,3,2,5,1,1,3,1,1,0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,2,5,9,8,1,0,5,8,8,5,2,5,4,6,0,9,1,2,3,5,0,5,1,3,3,5,7,2,3,1,0,0,2,3,5,8,5,7,8,3,4,0,8,7,2,6,7,9,7,9,8,7,3,6,5,7,6,7,1,9,3,3,6,9,4,6,2,6,8,3,5,8,6,4,6,3,4,7,8,6,1,2,9,7,4,9,3,5,1,7,4,4,3,2,5,1,1,3,1,1,0,4,2,4,9,7,9,2,4,2,1,8,9,0,6,7,0,1,0,4,8,7,1,2,1,4,9,6,3,0,7,8,7,5,0,1,2,9,6,9,0,2,4,0,5,8,9,6,8,0,2,5,9,8,1,0,5,8,8,5,2,5,4,6,0,9,1,2,3,5,0,5,1,3,3,5,7,2,3,1,0,0,2,3,5,8,5,7,8,3,4,0,8,7,2,6,7,9,7,9,8,7,3,6,5,7,6,7,1,9,3,3,6,9,4,6,2,6,8,3,5,8,6,4,6,3,4,7,8,6,1,2,9,7,4,9,3,5,1,7,4,4,3,2,5,1,1,3,1,1]

3. Getting familiar with the classifier modules and further metrics operations. Moreover, the histogram of oriented gradients is also studied.

4. Implementation of Logistic Regression Classifier.

5. Analysis of Classifier performance applying metrics.

6. Further experiments implementing the HOG function. Designation of new parameters, conversion of various images, with different noise factors, and analysis. A total of three images are compared and studied. The noise factors for these images are 1,5,10 correspondingly.

3.Approach / Resolution.

The very first step as expounded in the task section was creating the datasets. After having produced the digits, varying the font, but never altering the image size, then each digit is assigned to a specific dataset; afterwards, every set is then saved in a proper folder. The accessing path is then specified when reading the corresponding folder of every dataset.

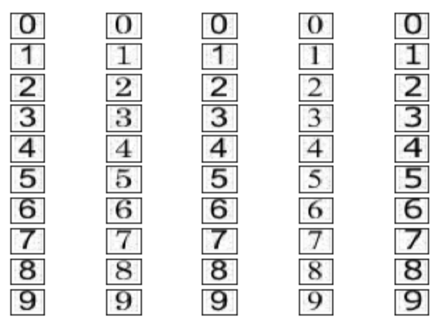


Figure 3. Datasets, X train, 50. Pattern, mixed fonts.

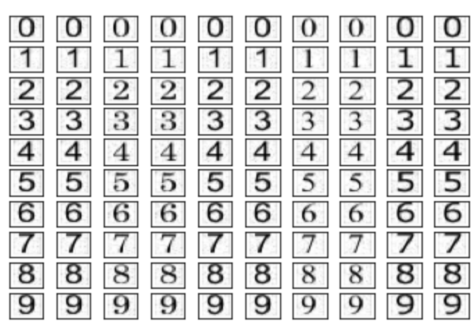


Figure 4. Datasets, X test, 100. Pattern, mixed fonts.

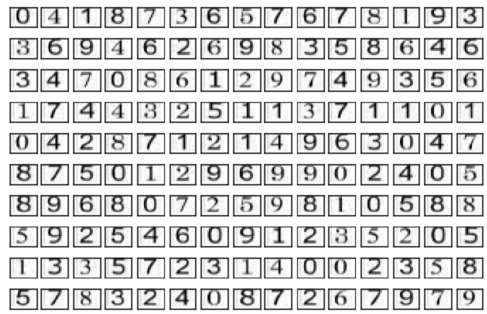


Figure 5. Dataset, X, 150 random, mixed fonts.

The implemented methods for inventing the classifier are then used:

1. Reading and plotting of folders, containing digits, using a for loop.
2. Conversion of read images to “numpy” arrays.
3. Reading of images as gray scale return a 2D array type. Since the implementation of sorting these images will be classified/fit to a certain label, and such label is 1D, then arrays are converted to 1D arrays as well.
4. During the first tests, a set of arrays, X_train,y_train, X_test, y_test, considering the step 3 guidelines, are properly assigned/created. For the second test, a dataset (X, y) is then created (step 3 guidelines), then it is split into X_train, X_train,y_train, X_test, y_test.
5. Finally for every case a logistic regression classifier and an estimator for classifier are used; as for data evaluation the classifier performance metrics as accuracy, fit score, precision, f1 score

Histogram Of Gradients

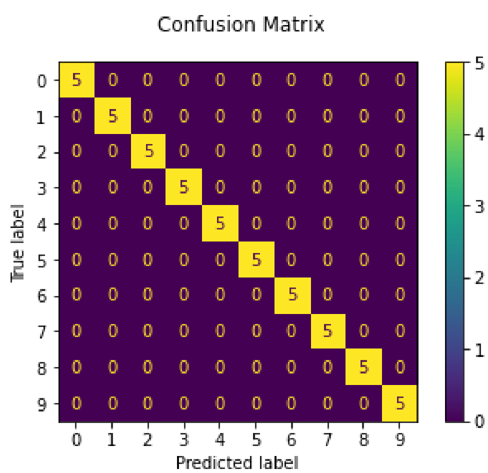
1. A particular image is read, the command or method imread from skimage.io is called.
2. For converting the selected image into a HOG, the function hog from skimage.feature executed.
3. The process of such conversion also includes an intensity rescaling since the conversion results in an “unclear image”.
4. These steps are repeated for the noise factor of 5, 10 images respectively.

4.Results and Discussion/ Conclusion.

Classifier Experiment. The first results from the first type classifier are shown in the following images:

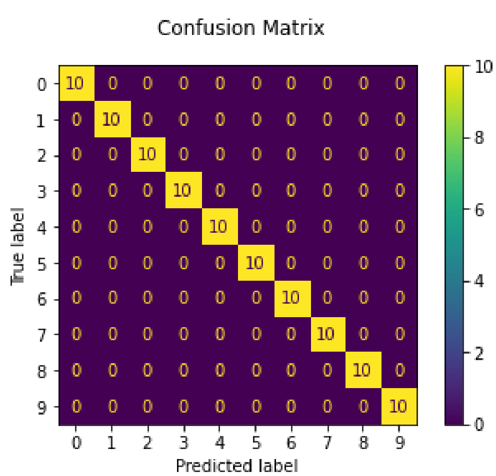
In this case, an intended array is converted into two different variables, X_train, X_test. These arrays are the proper read images, and converted, considering the former mentioned guidelines. As for the labels, y_train, y_test, are created manually in accordance with the images' values. Then, X_train is set equal to X_test, and these arrays have a particular sequence of 50 images, the results are as follows. A further different calculation is made by using a X_test of 100, yet same sequence of numbers is maintained.

Figure 6. Results Classifier 1.




```
Test=Train digits=50, mixed fonts
(50, 705600)
LogisticRegression score: 1.000000
accuracy:  1.0
Confusion matrix:
[[5 0 0 0 0 0 0 0 0 0]
 [0 5 0 0 0 0 0 0 0 0]
 [0 0 5 0 0 0 0 0 0 0]
 [0 0 0 5 0 0 0 0 0 0]
 [0 0 0 0 5 0 0 0 0 0]
 [0 0 0 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 5 0 0 0]
 [0 0 0 0 0 0 0 5 0 0]
 [0 0 0 0 0 0 0 0 5 0]
 [0 0 0 0 0 0 0 0 0 5]]
```

Figure 7. Results Classifier 1.1




```
Test=100 Train=50, mixed fonts
(100, 705600)
(50, 705600)
LogisticRegression score: 1.000000
accuracy:  1.0
Confusion matrix:
[[10  0  0  0  0  0  0  0  0  0]
 [ 0 10  0  0  0  0  0  0  0  0]
 [ 0  0 10  0  0  0  0  0  0  0]
 [ 0  0  0 10  0  0  0  0  0  0]
 [ 0  0  0  0 10  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 10  0  0  0]
 [ 0  0  0  0  0  0  0 10  0  0]
 [ 0  0  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0 10]]
```

The second test consisted of 600 random and mixed fonts digits. The main differences from the former and current case, is the fact of using solely one dataset, then split it into X_test, X_train, y_test, y_train using the command: `X_train, X_test, y_train, y_test = train_test_split(`

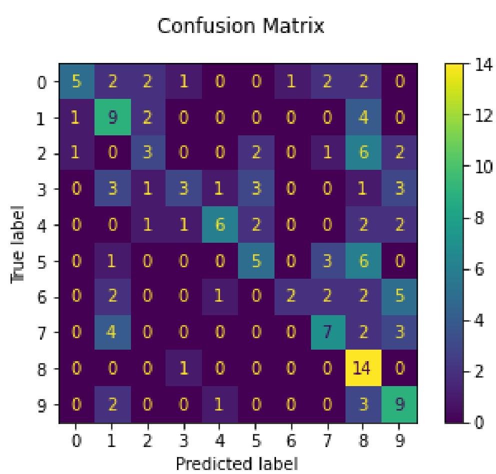
```
    X, y, test_size=0.5, random_state=None, shuffle=False, stratify=None)
```

*Note. The command entails a wider range of parameters, nonetheless these will not be explained into detail. In addition, for every case, a new value for a single or many parameter(s) is(are) modified.

Wherein X, y represents the current dataset and label respectively.

Prior to this attempt of classification a similar test was performed using 150 random and mixed fonts digits. For this case same procedure of X_test is set equal to X_train. These are the results.

Figure 8. X_test = X_train, 150 random mixed fonts




```
 Test=150 random sequence mixed font,
Train=150 random sequence mixed fonts
                       (150, 705600)
                       (150, 705600)
   LogisticRegression score: 0.420000
                       accuracy:  0.42
                     Confusion matrix:
 [[ 5  2  2  1  0  0  1  2  2  0]
  [ 1  9  2  0  0  0  0  0  4  0]
  [ 1  0  3  0  0  2  0  1  6  2]
  [ 0  3  1  3  1  3  0  0  1  3]
  [ 0  0  1  1  6  2  0  0  2  2]
  [ 0  1  0  0  0  5  0  3  6  0]
  [ 0  2  0  0  1  0  2  2  2  5]
  [ 0  4  0  0  0  0  0  7  2  3]
  [ 0  0  0  1  0  0  0  0 14  0]
  [ 0  2  0  0  1  0  0  0  3  9]]
```

600 random mixed fonts digits. Since for this very case the manner of creating a dataset and splitting it into different variables differs from the above cases, a variety of parameters is offered as well as a grater number of metrics, which are implemented for evaluating the performance of the current classifier.

Result 3.1. Test size = .90. `X_train, X_test, y_train, y_test = train_test_split(`

```
    X, y, test_size=0.9, random_state=None, shuffle=True, stratify=None)
```

```
(600, 705600) test.90
LogisticRegression score: 0.327778
Classification report for classifier SVC(gamma=0.0001):
               precision    recall  f1-score   support
```

```
              0       0.18      0.68      0.28        50
              1       0.54      0.47      0.50        58
              2       0.35      0.42      0.38        53
              3       0.00      0.00      0.00        59
              4       0.28      0.24      0.26        50
              5       0.39      0.24      0.30        55
              6       0.30      0.06      0.10        52
              7       0.50      0.17      0.25        59
              8       0.43      0.38      0.40        52
              9       0.33      0.54      0.41        52

       accuracy                           0.31       540
      macro avg       0.33      0.32      0.29       540
   weighted avg       0.33      0.31      0.29       540


accuracy:  0.31296296296296294
Confusion matrix:
[[34  4  8  0  0  4  0  0  0  0]
 [16 27  0  0  8  0  0  0  3  4]
 [20  0 22  0  0  0  0  0  0 11]
 [28  0  4  0  4  8  7  0  0  8]
 [16  4  7  0 12  0  0  4  0  7]
 [24  0  3  0  0 13  0  3  4  8]
 [12  4  7  0  7  8  3  3  4  4]
 [19  7  4  0  4  0  0 10  8  7]
 [13  4  4  0  4  0  0  0 20  7]
 [ 8  0  4  0  4  0  0  0  8 28]]
```

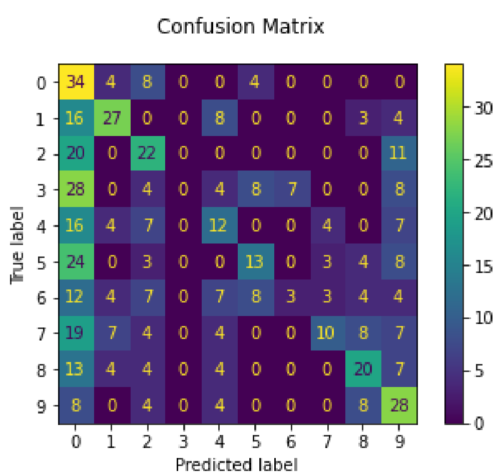


Figure 9. Results of test 3.1.

Result 3.2. Test size = .33. `X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.33, random_state=None, shuffle=False, stratify=None)`

```
(600, 705600)
Classification report for classifier SVC(gamma=0.0001):
              precision    recall  f1-score   support

           0       0.29      0.80      0.43        15
           1       0.59      0.57      0.58        23
           2       0.33      0.28      0.30        18
           3       0.54      0.30      0.39        23
           4       0.23      0.15      0.18        20
           5       0.50      0.32      0.39        19
           6       0.00      0.00      0.00        22
           7       0.32      0.67      0.43        21
           8       0.71      0.28      0.40        18
           9       0.29      0.47      0.36        19

    accuracy                           0.37       198
   macro avg       0.38      0.38      0.35       198
weighted avg       0.38      0.37      0.34       198


accuracy:  0.37373737373737376
Confusion matrix:
[[12  0  1  0  0  0  0  2  0  0]
 [ 4 13  0  0  0  0  0  3  0  3]
 [ 2  0  5  0  0  2  0  5  0  4]
 [ 4  2  4  7  2  0  0  2  0  2]
 [ 3  1  1  2  3  0  0  7  0  3]
 [ 4  0  2  1  0  6  0  1  0  5]
 [ 4  4  1  3  3  2  0  3  1  1]
 [ 0  2  0  0  2  2  0 14  1  0]
 [ 4  0  1  0  1  0  0  3  5  4]
 [ 4  0  0  0  2  0  0  4  0  9]]
```

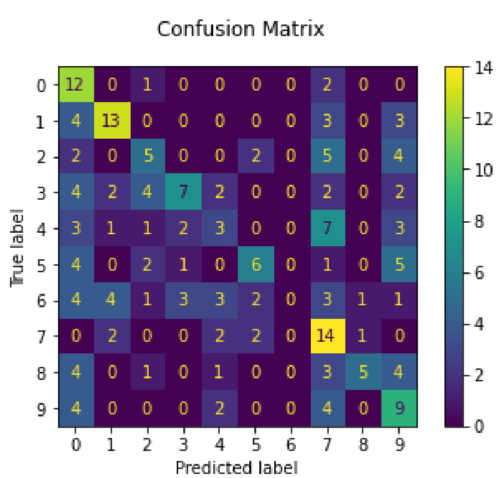


Figure 10. Results of test 3.2.

An additional Test is made using the corresponding parameters:

Result 3.3. Test size = .75. `X_train, X_test, y_train, y_test = train_test_split(`
`X, y, test_size=0.33, random_state=None, shuffle=True, stratify=None)`

```
(600, 705600)
LogisticRegression score: 0.426667
Classification report for classifier SVC(gamma=0.0001):
              precision    recall  f1-score   support

           0       0.38      0.40      0.39        45
           1       0.73      0.50      0.59        48
           2       0.42      0.33      0.37        45
           3       0.67      0.27      0.38        45
           4       0.33      0.21      0.26        42
           5       0.67      0.27      0.38        45
           6       0.33      0.29      0.31        42
           7       0.37      0.88      0.52        48
           8       0.67      0.27      0.38        45
           9       0.35      0.80      0.49        45

    accuracy                           0.43       450
   macro avg       0.49      0.42      0.41       450
weighted avg       0.49      0.43      0.41       450

accuracy:  0.4266666666666667
Confusion matrix:
[[18  0  6  0  6  0  3  6  0  6]
 [ 6 24  3  0  0  0  3  6  0  6]
 [ 3  0 15  0  3  3  0  9  0 12]
 [ 6  0  3 12  3  0  9  6  0  6]
 [ 0  3  0  0  9  0  6 15  0  9]
 [ 6  0  3  3  3 12  3  6  0  9]
 [ 0  3  3  3  0  3 12  9  3  6]
 [ 0  3  0  0  0  0  0 42  3  0]
 [ 9  0  3  0  3  0  0  6 12 12]
 [ 0  0  0  0  0  0  0  9  0 36]]
```

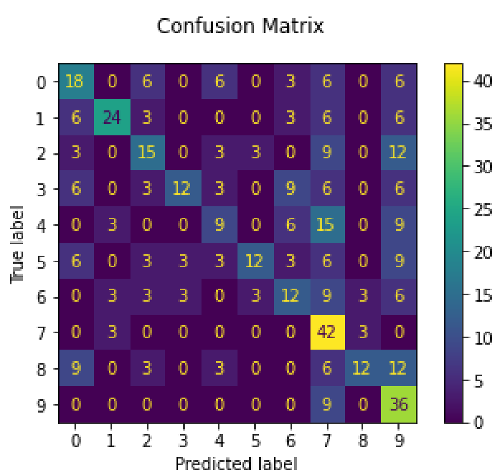


Figure 11. Results of test 3.3.

Discussion/ Conclusion: The logistic regression function is given by the following command.

```
logistic = linear_model.LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='ovr', n_jobs=None, penalty='l2', random_state=0,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

Most of the parameters are set to default values in this experiment. Nonetheless, for a more complexed and particular cases, these parameters tend to be altered as to fulfill and yield better results. Within these parameters one can change the penalty form, the type of solver, max number of iteration and so on.

When using the customized form of assigning values to X_test, X_train, y_test, y_train, the classifier tended to have a very good accuracy. Nonetheless, the limitation was that a set with similar sequence, or pattern in the digits must be used. When such pattern or sequence is similar to that of X_train=X_test, no matter the size of the test set, it mostly proved to return a perfect prediction, accuracy (1.0). In the other hand, when the command for splitting a sole dataset was put into practice, the accuracy and prediction value were not that good. The best obtained result was 0.42. Precision was also noted, it can be concluded that such classifier lacks certain level of precision. For some digits this value was above 0.50, whereas for the rest below this number.

In contrast to the first type of make a classifier, it was also experienced a high processing time for running the code, since the dataset was considerably greater in size and more data to be obtained, the PC took up to 25 minutes to finalize the computing of such classifier. As a consequence of the latter, the model was sometimes converting data to 0, due to lack of capability for recognizing the data. When this occurred, the samples were assumed to be 0, but were not taken into consideration when evaluating the classifier performance. A curios fact was that whenever the test set was near to twice the size of the training set, the results were “better”/ “increased”.

PC INFO.

Device name	LAPTOP-LPTB78ML
Processor	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
Installed RAM	24.0 GB (23.8 GB usable)
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

The proposed solution for future applications is to strengthen the classifier by implementing and altering the different parameters within such mentioned command. In addition, a better dataset with greater data and “specific” sequence/ design could be particularized, for a specific application. Nonetheless, the hardware limitations shall be considered as a crucial matter as well. As a result, one could develop a more robust classifier model.

HOG Experiment. For attaining the resolution of this experiment, the command hog is implemented.

```
fd, hog_image = hog(im0_times, orientations=8, pixels_per_cell=(24, 24),
                    cells_per_block=(1, 1), visualize=True)
```

As expounded before, the reading of any image can be done by the command imread from the module/ package skimage.io. A total of three images are read, the font in use is Times, Century. First image has a noise value of 1, second image a noise value of 5, and lastly third image a noise value of 10.

Figure 12. Digits with different noise.

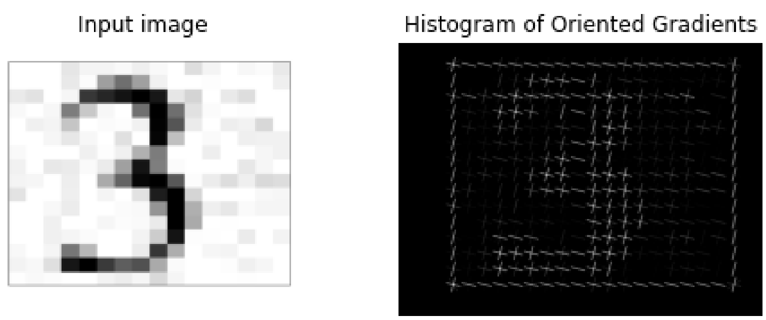


Figure 13. Digit 1 converted to HOG.

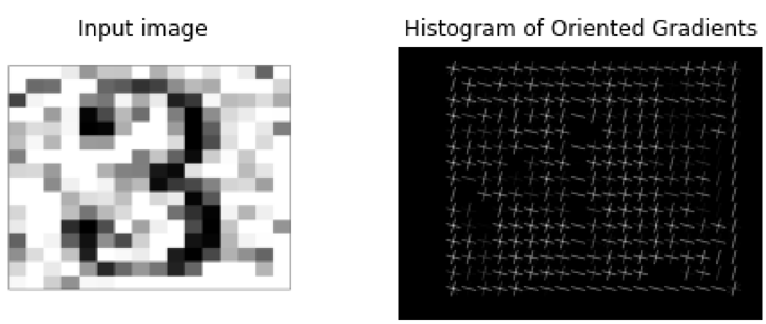


Figure 14. Digit 2 converted to HOG

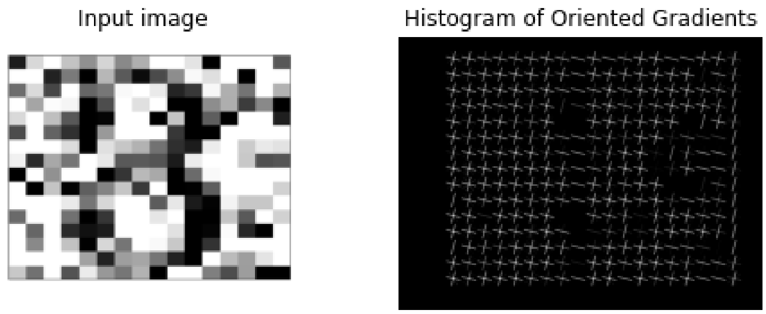


Figure 15. Digit 3 converted to HOG.

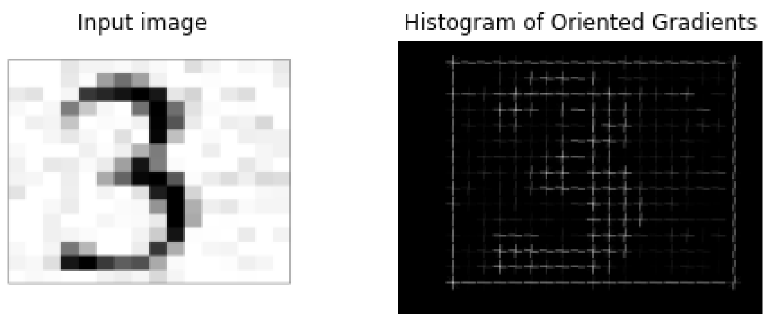


Figure 16. Digit 1 with 32 orientations, cells 24,24.

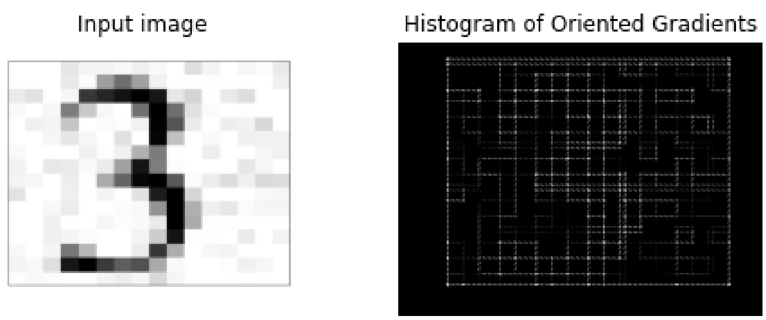


Figure 17. Digit 1 with 24 orientations, Cells 8,8.

Discussion/ Conclusion. The comparison between the noised digits, as higher the noise factor is, the higher the difficulty to read (visualize) and distinguish the digit. When varying the number of orientations and cells, one could either expand the number of pixels within every cell resulting in less displayed gradients (number of cells within image for representing a gradient is reduced), or number of orientations resulting in a more "detailed" image representation (in gradients). The second parameter is to be modified when more details are within an image specifically, and one intends to display these, but also one must consider the value of numbers of cells. The latter changing drastically the output number of gradients displayed in any image and may overlook certain regions of such converted image.

5. References.

Works Cited

[1] S. Dey, Image Processing MasterClass with Python, New Delhi: Manish Jain for BPB Publications, 2021.

[2] N. D. a. B. Triggs, "Histograms of Oriented Gradients for Human Detection," p. 8.

[3] B. Klein, Machine Learning with Python Tutorial, 2021.

[4] J. D. IBM, "IBM.com," International Business Machine Corporation, [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

6. Appendix (Python's Code)

Reading folder from path and importing of site Packages.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec  7 22:03:52 2021

@author: jlqgj
"""

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import numpy as np
from skimage import data, img_as_float
from skimage import exposure
from skimage.io import imread, imsave, imshow
from skimage.feature import hog
from skimage import data, exposure
from skimage.util import img_as_ubyte
import os
from sklearn.metrics import classification_report, accuracy_score
from skimage.io import imread, imsave, imshow
import glob
import cv2
from sklearn import svm, metrics
from sklearn.model_selection import train_test_split
from sklearn import neighbors, linear_model


#Reading/Plotting digits=150 font=mixed scattered
#Mixed fonts
im_list_mixed150scattered = [cv2.imread(file) for file in
glob.glob('C:/Users/jlqgj/numbers/150digitsscattered/*.png')]
for i in range(150):
    plt.subplot(10,15,i+1),plt.imshow(im_list_mixed150scattered[i],'gray')
```

```
    plt.xticks([]),plt.yticks([])
plt.show()
```

There are two manners of comparing a X Training set with a X Testing set. Either one can create an array for each of these variables, including the corresponding label respectively, or one can create a data set then split it into training and testing sets. The latter option is easier but offers certain flexibility. The data and other sets can be shuffled, for instance. Nonetheless, it does not offer a customized training or testing set, or data arrangement, when some particularizations are desired.

First Manner of creating Data, Training, Testing Set.

```
im_list_600mixedrandom = [cv2.imread(file) for file in
glob.glob('C:/Users/jlqgj/numbers/600mixedrandom/*.png')]
mixedrandom600_font = np.asarray(im_list_600mixedrandom)
n_samples_600mixedrandom = len(mixedrandom600_font)
mixedrandom600_rs = mixedrandom600_font.reshape((n_samples_600mixedrandom,-1))
X = mixedrandom600_rs
print(X.shape)

y = [0,4,2,4,9,7,9,2,4,2,1,8,
            9,0,6,7,0,1,0,4,8,7,1,2,
            1,4,9,6,3,0,7,8,7,5,0,1,
            2,9,6,9,0,2,4,0,5,8,9,6,
            8,0,2,5,9,8,1,0,5,8,8,5,
            2,5,4,6,0,9,1,2,3,5,0,5,
            1,3,3,5,7,2,3,1,0,0,2,3,
            5,8,5,7,8,3,4,0,8,7,2,6,
            7,9,7,9,8,7,3,6,5,7,6,7,
            1,9,3,3,6,9,4,6,2,6,8,3,
            5,8,6,4,6,3,4,7,8,6,1,2,
            9,7,4,9,3,5,1,7,4,4,3,2,
            5,1,1,3,1,1,0,4,2,4,9,7,
            9,2,4,2,1,8,9,0,6,7,0,1,
            0,4,8,7,1,2,
            1,4,9,6,3,0,7,8,7,5,0,1,
            2,9,6,9,0,2,4,0,5,8,9,6,
            8,0,2,5,9,8,1,0,5,8,8,5,
            2,5,4,6,0,9,1,2,3,5,0,5,
            1,3,3,5,7,2,3,1,0,0,2,3,
            5,8,5,7,8,3,4,0,8,7,2,6,
            7,9,7,9,8,7,3,6,5,7,6,7,
            1,9,3,3,6,9,4,6,2,6,8,3,
            5,8,6,4,6,3,4,7,8,6,1,2,
            9,7,4,9,3,5,1,7,4,4,3,2,
            5,1,1,3,1,1,0,4,2,4,9,7,9,2,4,2,1,8,
            9,0,6,7,0,1,0,4,8,7,1,2,
            1,4,9,6,3,0,7,8,7,5,0,1,
            2,9,6,9,0,2,4,0,5,8,9,6,
            8,0,2,5,9,8,1,0,5,8,8,5,
            2,5,4,6,0,9,1,2,3,5,0,5,
            1,3,3,5,7,2,3,1,0,0,2,3,
            5,8,5,7,8,3,4,0,8,7,2,6,
            7,9,7,9,8,7,3,6,5,7,6,7,
            1,9,3,3,6,9,4,6,2,6,8,3,
            5,8,6,4,6,3,4,7,8,6,1,2,
            9,7,4,9,3,5,1,7,4,4,3,2,
            5,1,1,3,1,1,0,4,2,4,9,7,9,2,4,2,1,8,
            9,0,6,7,0,1,0,4,8,7,1,2,
```

```
            1,4,9,6,3,0,7,8,7,5,0,1,
            2,9,6,9,0,2,4,0,5,8,9,6,
            8,0,2,5,9,8,1,0,5,8,8,5,
            2,5,4,6,0,9,1,2,3,5,0,5,
            1,3,3,5,7,2,3,1,0,0,2,3,
            5,8,5,7,8,3,4,0,8,7,2,6,
            7,9,7,9,8,7,3,6,5,7,6,7,
            1,9,3,3,6,9,4,6,2,6,8,3,
            5,8,6,4,6,3,4,7,8,6,1,2,
            9,7,4,9,3,5,1,7,4,4,3,2,
            5,1,1,3,1,1]


#Solely when using a same data set for creating testing and training set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.66, random_state=None, shuffle=False, stratify=None)
```

Second Manner of creating Data, Training, Testing Set. (Customized)

```
# Convert list to numpy array
# Test Data
mixed50_font = np.asarray(im_list_mixed50)
n_samples_mixed50 = len(mixed50_font)
mixed50_rs = mixed50_font.reshape((n_samples_mixed50,-1))
X_train = mixed50_rs
print(X_train.shape)
y_train = [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,6,6,6,
             6,6,7,7,7,7,7,8,8,8,8,8,9,9,9,9,9]

# Convert list to numpy array
# Test Data mixed 100
mixed100_font = np.asarray(im_list_mixed100)
n_samples_mixed100 = len(mixed100_font)
mixed100_rs = mixed100_font.reshape((n_samples_mixed100,-1))
X_train = mixed100_rs
print(X_train.shape)
y_train = [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,
            2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,
            4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,
            6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,
            8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9]

# Convert list to numpy array
# Test Data
mixed150_fontscattered = np.asarray(im_list_mixed150scattered)
n_samples_mixed150scattered = len(mixed150_fontscattered)
mixed150scattered_rs = mixed150_fontscattered.reshape((n_samples_mixed150scattered,-1))
X_test = mixed150scattered_rs
print(X_test.shape)
y_test = [0,4,2,4,9,7,9,2,4,2,1,8,
             9,0,6,7,0,1,0,4,8,7,1,2,
             1,4,9,6,3,0,7,8,7,5,0,1,
             2,9,6,9,0,2,4,0,5,8,9,6,
             8,0,2,5,9,8,1,0,5,8,8,5,
             2,5,4,6,0,9,1,2,3,5,0,5,
             1,3,3,5,7,2,3,1,0,0,2,3,
             5,8,5,7,8,3,4,0,8,7,2,6,
             7,9,7,9,8,7,3,6,5,7,6,7,
             1,9,3,3,6,9,4,6,2,6,8,3,
             5,8,6,4,6,3,4,7,8,6,1,2,
             9,7,4,9,3,5,1,7,4,4,3,2,
             5,1,1,3,1,1]
```

Evaluating, fitting, Classifying Data

```
clf = svm.SVC(gamma=0.001)

logistic = linear_model.LogisticRegression(solver='sag',max_iter=10000)

#print("KNN score: %f" % knn.fit(X_train, y_train).score(X_test, y_test))
print(
    "LogisticRegression score: %f"
    % logistic.fit(X_train, y_train).score(X_test, y_test)
)

clf.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = clf.predict(X_test)

print(
    f"Classification report for classifier {clf}:\n"
    f"{metrics.classification_report(y_test, predicted)}\n"
)
print('accuracy: ', accuracy_score(y_test, predicted))

disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, predicted)
disp.figure_.suptitle("Confusion Matrix")
print(f"Confusion matrix:\n{disp.confusion_matrix}")

plt.show()
```

Histogram Of Oriented Gradients

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import numpy as np
from skimage import data, img_as_float
from skimage import exposure
from skimage.io import imread, imsave, imshow
from skimage.feature import hog
from skimage import data, exposure
from skimage.util import img_as_ubyte
import os
from sklearn.metrics import classification_report, accuracy_score

from skimage.io import imread, imsave, imshow
import glob
import cv2

im_dict_times_1 = [cv2.imread(file) for file in glob.glob('C:/Users/jlqgj/numbers/Times/*.png')]
label_1 = ['zero','one','two','three','four','five','six','seven','eight','nine']
for i in range(10):
    plt.subplot(3,4,i+1),plt.imshow(im_dict_times_1[i],'gray')
    plt.title(label_1[i], fontsize=9)
    plt.xticks([]),plt.yticks([])
plt.show()

im0_times = im_dict_times_1[0]
fd, hog_image = hog(im0_times, orientations=8, pixels_per_cell=(24, 24),
                    cells_per_block=(1, 1), visualize=True)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

ax1.axis('off')
ax1.imshow(im0_times, cmap=plt.cm.gray)
ax1.set_title('Input image')
```

```
# Rescale histogram for better display
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 5))

ax2.axis('off')
ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```