Classification of handwritten digits using MNIST dataset

1. Motivation:

Modified National Institute of Standards and Technology abbreviated as MNIST is an extensive collection of hand-written digits in the range of 0 to 9. It is the go-to dataset for beginners who want to explore image processing. MNIST has a collection of 70,000, 28 x 28 images of handwritten digits. The creator's dataset is available on the site, consisting of training set images of 9912422 bytes, training set labels of 28881 bytes, test set images of 1648877 bytes and test set labels of 4542 bytes. This raw dataset consists of images and labels but we can directly import the dataset from the Keras or SKlearn module. Hence, with this dataset, I aim to create a Classification model that will correctly classify digits that are in handwritten format. One of the features of Mac OS X that intrigued me is how my machine is able to detect my handwriting and allows me to copy-paste it in text format, inspired by this I was motivated to create my project which follows a similar process.

2. Prior Work:

1. MNIST digit recognition using SVM by nishan192 on Kaggle
2. Machine-Learning-Projects- SVM with MNIST by dmkothari

In the above-mentioned MNIST projects, only one SVM classifier is built, I use the dataset to fit an SVM model as well as a Logistic Regression model. In a way, I am using the MNIST dataset for benchmarking machine learning models - Support Vector Machine and Logistic Regression.

3. Model

Support Vector Machine

- **About the MNIST dataset:**

The MNIST dataset is imported from sklearn.datasets. The load_digits() function loads and returns the digits dataset for classification. We see that the data has keys such as 'data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR' in the dictionary. With ‘train_test_split’ we split the dataset into a testing set of 25% and a training set of 75%. In the

‘images’ category we have pixel images of each number ranging from 0 to 9. We can visualize this using Matplotlib. As the first row represents the digit ‘0’, we see a grey pixelated image of the handwritten digit ‘0’ when the first row is plotted.

- **Building the SVM model:**

SVM, a supervised learning algorithm that can be used for classification tasks can easily build the SVM model using Sklearn. The SVC class in scikit-learn's SVM module stands for Support Vector Classification. To use the SVC class, we instantiate an SVC object and ‘model = SVC()’ builds a new SVC object and assigns it to the model variable.

- **Cross-validation using GridSearchCV:**

Our next task is to use the fit method on the model object and pass it to the training data and labels but before that, we perform cross-validation. GridSearchCV is a technique for finding the optimal parameter values from a given set of parameters in a grid. GridSearchCV will perform an exhaustive search over a specified parameter grid for an estimator. It will train as well as evaluate the estimator for each combination of the specified parameters and return the combination which gives the best performance according to a scoring function. In the project, the ‘model’ is the support vector machine classifier (estimator), and ‘params’ is a dictionary that defines the parameter grid to search over. The keys of the dictionary- ‘C' and ‘gamma’ are the names of the parameters and the values are lists of the possible values for each parameter. The `GridSearchCV` function will evaluate the estimator using the following combinations of `C` and `gamma`: (1, 0.1), (1, 0.01), (1, 0.001), (10, 0.1), (10, 0.01), (10, 0.001), (100, 0.1), (100, 0.1), and (100, 0.001).

To use `GridSearchCV`, we call the `fit` method, passing it the training data and labels - X_train and y_train. This will trigger the search for the best combination of parameters. Once the search is complete, we access the best combination of parameters and the best estimator.

- **Attributes displayed:**

1. **best_score_:** An attribute will return the best mean cross-validated score achieved across all the parameter combinations in the grid search. This score is usually the score of the best model.

2. **best_estimator_:** We want the best model to make predictions on new data. Hence, we use this attribute, which will return the best model found by the grid search, with the optimal set of hyperparameters.
3. **best_params_:** We want to know which hyperparameters resulted in the best model. Hence, this attribute is used which returns the best hyperparameter combination found by the grid search.
4. Finally, the **'score'** method of a GridSearchCV object returns the mean accuracy on the given test data and labels. It applies the best model found by the grid search to the test data and returns the mean accuracy of the predictions made by the model.

- **Testing on real-time data apart from the dataset:**

Consider the image below which is not a part of the dataset and is unique, our task is to predict and identify these handwritten digits, we can do it with the help of OpenCV library

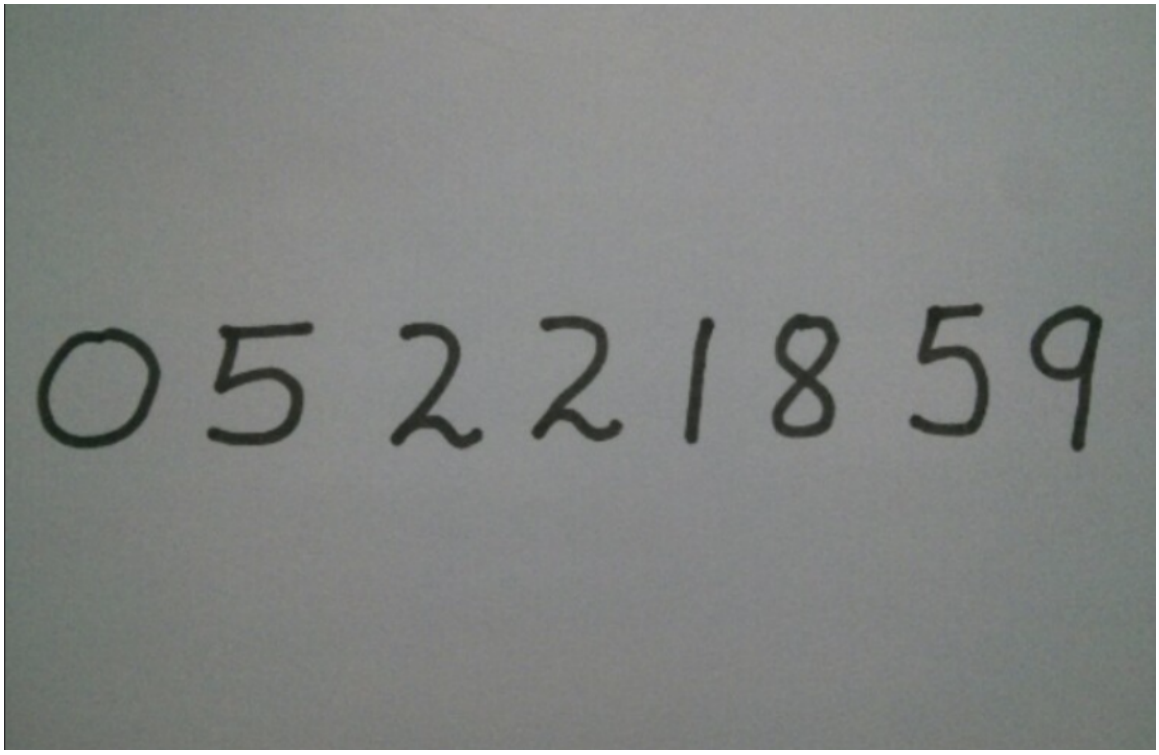


Figure 1: *Handwritten numbers in a random order, Source: Geeksforgeeks, Blog Topic: Tesseract OCR with java with examples*

- The ‘**cv2.cvtColor’** function from the OpenCV library converts an image from one colour space to another. It took our image input and a colour conversion code and returned the converted image. As we are only interested in the intensity values of the pixels and not the colour information we use the cv2.cvtColor function
- The ‘**cv2.COLOR_BGR2GRAY’** is the colour conversion code that specifies that the image should be converted from the BGR colour space to the grayscale colour space.
- The ‘**cv2.threshold’** function from OpenCV library applies a certain threshold to an image. It takes the input as follows - an image, a threshold value, thresholding type and returns a thresholded image. The ‘grey.copy()’ function creates a copy of the grey image, which is then passed as the first argument to cv2.threshold. This is done to prevent modifying the original image, as the cv2.threshold function changes the input image in place.
- The ‘75’ parameter is the threshold value that specifies the intensity level above which the pixels in the image will be set to the maximum intensity value ‘255’. Pixels with intensity values below the threshold will be set to the minimum intensity value ‘0’.
- The ‘**cv2.THRESH_BINARY_INV’** parameter specifies the thresholding type, which in this case is binary inverse thresholding. This means that pixels above the threshold will be set to 0 and pixels below the threshold will be set to 255.
- The ‘**cv2.findContours’** function in the OpenCV library extracts the contours of objects in an image. The green coloured outlines are the contours in our case. It takes as input - image, a contour retrieval mode, and a contour approximation method, and returns a list of contours and a hierarchy of contours. thresh.copy() creates a copy of the thresh image, which is then passed as the first argument to cv2.findContours. This is done to prevent modifying the original image, as the cv2.findContours function changes the input image in place.
- The ‘**cv2.RETR_EXTERNAL’** parameter specifies the contour retrieval mode, which in this case is ‘cv2.RETR_EXTERNAL’. This means that only the outermost contours will be extracted and returned.
- The ‘**cv2.CHAIN_APPROX_SIMPLE’** parameter specifies the contour approximation method, which in this case is ‘cv2.CHAIN_APPROX_SIMPLE’. This means that the contours will be simplified by approximating the points on the contours with a single point.

Hence, the digits are recognized as follows:

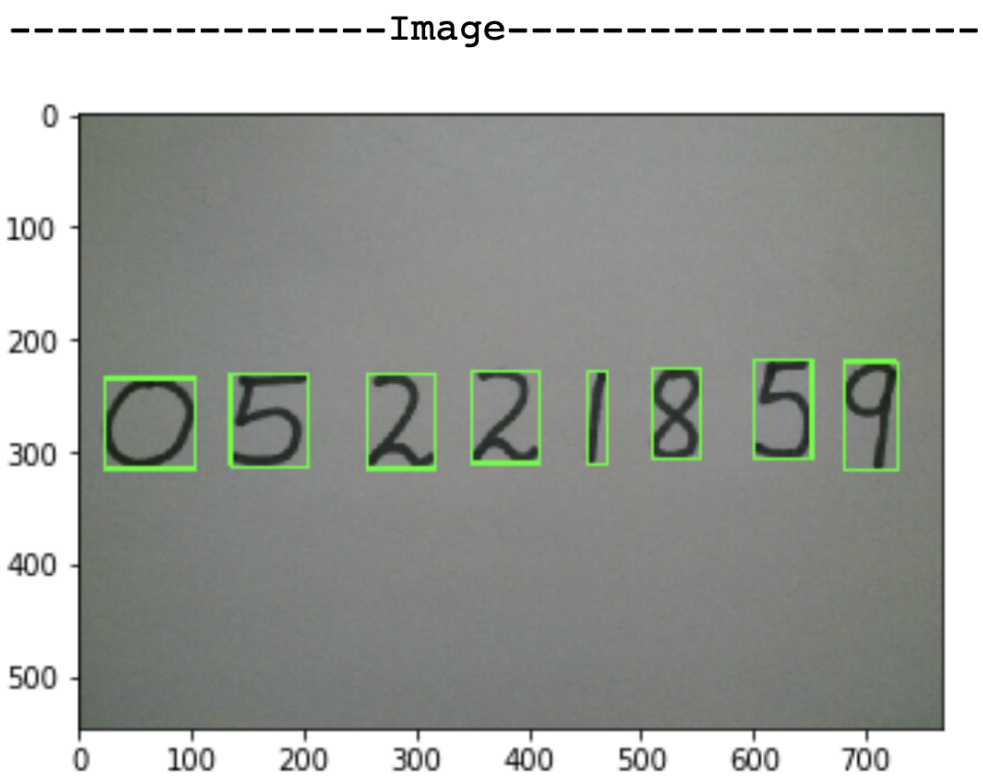
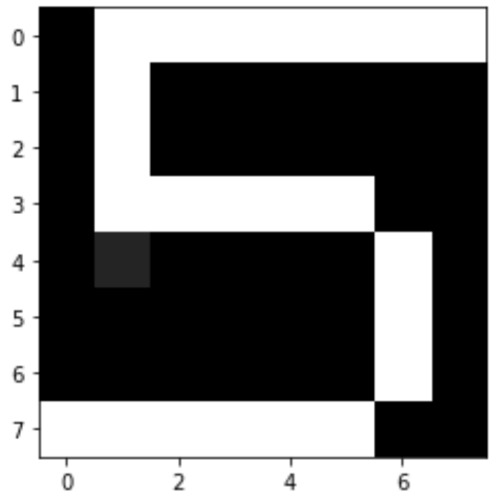


Figure: Recognized digits using OpenCV, Source: my project's i python notebook

Now, Prediction is done on all these newly recognized digits using 'model_linear.predict(digit)' and displayed accordingly. One example is as follows:

```
---------PREDICTION----------
```




```
(8, 8)
```

Figure: Prediction of the digit 5 from the real-time data , Source: my project's i python notebook

Logistic Regression

- **Building the Logistic Regression model:**

After importing the data and splitting it into the train (70%) and test sets (30%) like we did with the SVM model. We create a logistic regression model using LogisticRegression()

- **Cross-validation to evaluate a model using k-fold cross-validation :**

The 'cross_val_score' is a function in the scikit-learn library that can be used to evaluate a model using k-fold cross-validation. It takes as input a model, the training data, the labels, and the number of folds, and returns an array of scores for each fold of the cross-validation. 'cv=5' specifies that the cross-validation should be performed using 5 folds. 'lr' is the model that we want to evaluate, 'X1_train' and 'y1_train' are the training data and labels

The 'scores' variable will contain an array of scores for each fold of the cross-validation. We use these scores to assess the performance of the model on the training data.

A 5-fold cross-validation on the 'lr' model using the training data and labels is performed and p the mean of the scores is computed. With, the help of the mean we can see how well the model is performing on our training data and how consistent its performance is across different folds.

The fit method of a model in scikit-learn is used to train the model on a given dataset. In this case, the model is being trained on the 'X1_train' and 'y1_train' data.

After the model has been trained, the predict method is used to make predictions on new data. In our project, the model is making predictions on the ‘X1_test’ data and storing the predictions in the ‘y1_pred’ variable.

The score method is used to evaluate the model. It returns the mean accuracy of the model on the data. In this case, the model is being evaluated on the X1_test and y1_test data.

4. Results and findings

1. For the SVM model using GridSearchCV: The best score across all searched parameters are 0.985894580549369
2. The best estimator across ALL searched params is given as SVC (C=1, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)
3. The best parameters across ALL searched params: {'C': 1, 'gamma': 0.001}
4. For SVM model, When evaluating the model on the test set, we get the Acurracy = 0.9888888888888889
5. For Logistic Regression model, after using cross-validation for 5 folds: When evaluating the model on the test set, we get the Acurracy = 0.9196666666666666
6. Hence, SVM performs better for this dataset as compared to Logistic Regression