Maame Nyarko
December 2, 2018

Exploring Handwritten Digits by loading and Visualizing the digits image data

The problem tackled in this project is known as the Classification of digits problem. This problem is an optical character recognition problem where handwritten digits are identified. The data was loaded from the in-built dataset in scikit learn which is called load_digits. The plt function from Matplotlib was then called to plot the digits on the screen together with their corresponding labels. The images data is a large array containing a lot of samples, so the sample limit chosen for this project is 100. Each of these images consist of an 8 x 8 grid of pixels. Three models were applied to the dataset: The Gaussian Naive-Bayes model, The Support Vector Machines and the Random Forest Classification model.

First, The Gaussian Naive-Bayes model. The dataset was first plotted to visualize the data before predictions were made for it. It was visualized with attributes such as color which would show as green for the digit corresponding to each image. Also the jupyter notebook had to be prompted to show the plot using “plt.show()” as this specifically told jupyter to display the plot. Without this command nothing was displayed. The model Hyperparameters used were Xtrain, Xtest, ytrain, ytest and they were set to train_test_split which is a module imported from scikit learn’s cross_validation. Train_test_split contained X, y which are the variables containing the data and the target. These Hyperparameters were used for the SVC and Random Forest models as well. After the Gaussian Naive-Bayes model was created, the hyperparameters Xtrain and ytrain were fit into it. The model was then asked to predict Xtest and store it in a parameter called y_model. Other modules used were the confusion_matrix module and the accuracy_score module imported from scikit learn’s metrics library. The confusion matrix helped to visualize the number of times the right digit was predicted for the right image and the number of times the model confused the digit as another digit. The accuracy score was then calculated by checking the average percentage of correctness for all digits. The predicted data was then plotted to show which images were predicted wrongly. If an image was predicted wrongly the color of the digit label changed to red. The accuracy score for the Gaussian Naive-Bayes model was calculated as 83%. Which is the lowest among the three models used. This process was pretty straightforward and could not be mistaken as complicated as compared to the other models used.

Second, the Support Vector Machine model. Here numpy was used with scikit learn and matplotlib. First the initial dataset was plotted just like it was in the Gaussian model. Then X and y were set to the data and the target. The Kernel Support Vector Machine was then plotted in 3D to be able to clearly visualize how the data and their target were clearly separated. The 3D plot was set to be viewed from the side to allow the reader to clearly see all points displayed from top to bottom. The 3D plot has an elevation attribute which can be changed to suit the reader’s perspective and the azim attribute which has been fixed as a side view and cannot be changed. The SVC and PCA functions were then called to be used to make a pipeline for the model. The PCA function has the attributes svd_solver which is set to “randomized”, n_components which is set to 64 since the size of the data set is 64 and the random state is set to 9. The SVC function has the attribute kernel which is set to “rbf” since the the dataset is non-linear, and the class weight is set to be balanced.

After the the pipeline has been made for the model all the hyperparameters needed for the data and target are called and pointed to train_test_split for cross_validation. The module GridSearchCV is used and the parameters for the grid which are “svc_C” and “svc__gamma” are set to range between “1,5,10” and “0.0001, 0.0005, 0.001” respectively. The Function GridSearchCV is the passed to the model and the parameters and stored in the grid variable.

To display the time taken for the model to train the data, the hyperparameters Xtrain and ytrain are fit into the grid and the best parameters are printed out. The output contains the time taken for the user to display the data and the system to display the data as wells as the points used from the svc range.
The model then predicts the data and stores it into the yfit hyperparameter and the yfit hyperparameter is then used together with Xtest parameter to plot the new dataset from the prediction made. Xtest is set as the data and yfit is set as the target. Just like the other models, this model also has a calculated confusion metrics plot.
The average precision, recall and f1-score for this model was 99% which was the highest accuracy score among the three models used. For a while there was a bit of uncertainty as to whether this score score was accurate. The total number of support was 450 which is equal to the total number of support used for the Random Forest model so this was a sign that the accuracy score could be correct.

Finally, the Random forest Classifier model. Once data set was plotted, the data and target were set just like the other models. Here NumPy, matplotlib, scikit learn and seaborn were used. The hyperparameters were set to the train_test_split function. Then the Random Forest Classifier model was created with an n_estimators attribute with the value of 1000. The hyperparameters Xtrain, ytrain were fit into the model and the model was then asked to predict the Xtest test data and store it into the ypred prediction parameter. A classification report was also made for this model using ypred and ytest. The average precision, recall and f1-score for this model was 98% which was the second highest and the support was also 450. The confusion matrix was also pretty similar to the Support Vector Machine model.

In conclusion, the Random Forest Classification model was expected to have the highest accuracy but surprisingly the Support Vector Machine did. Though the plots for their predicted datasets showed that the Random Forest Classification model had no red colored digits to indicate a wrong prediction while the Support Vector Machine had one red colored digit. Also, the digit 9 had the lowest accuracy on all counts of precision, recall, and f1-score at 95% which was lower than the accuracy score for the digit 9 in the Random Forest Classifier which was at 98%. The only advantage the Support Vector Machine had was that it had a precision of 100% for more than half of the digits
Using the Support Vector Machine caused the runtime to be slower and it was more difficult to determine if the performance data was correct or not. This is because fitting the parameters into the model is not done through the SVM dataset but through a pipeline so visualizing the process is more difficult and passing the data through all those channels makes it more difficult for the performance data to be accurate. This is why the Random Forest Classifier is the most optimal. It is just as straightforward as the Gaussian Naive-Bayes model and it also has a very high performance.
The interesting part of this project is that these steps created can be reused for solving different problems.

	Gaussian Naive-Bayes	Support Vector Machine	Random Forest Classifier
Precision	87%	99%	98%
f1-score	83%	99%	98%
Average accuracy	83%	99%	98%