Kaggle Digit Image Classification

I. Introduction

This report covers the practical implementation of Supervised Machine Learning technique named Classification [1] using different techniques on Kaggle [2] digit image dataset. We will be covering three different algorithms of Classification [3] as Naive Bayes, Support Vector Classifiers and K Nearest Neighbors. Dataset contains 1400 random images of hand written images of (28*28) pixels grid numbered from 0 to 9 as shown in Figure-1.

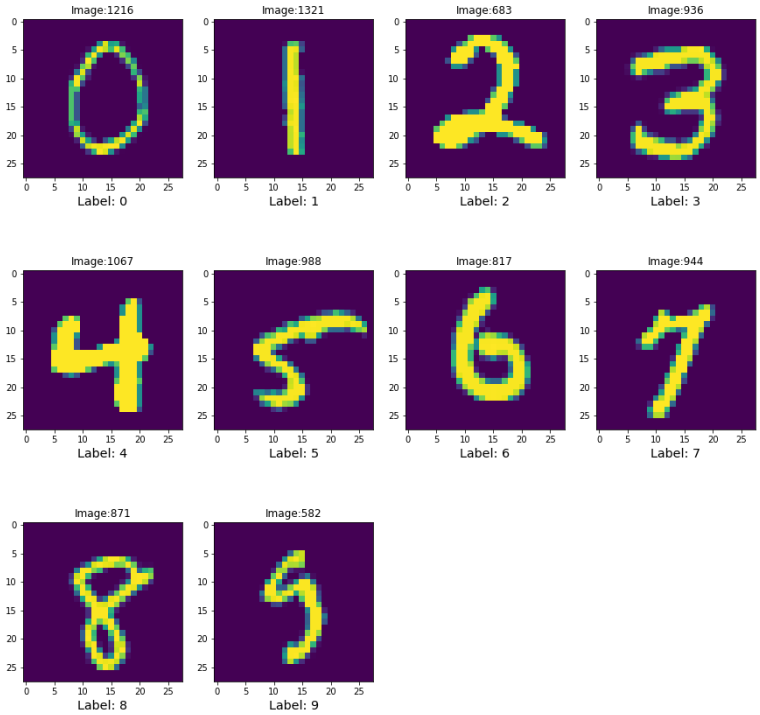

Figure 1: Depiction of Digits

Our main goal for this task is to classify the digits correcty as precisely as we can using the given classification techniques. Before applying these techniques we have to look for impurities in data such as missing values, duplicates etc and then we have to process the data. We have looked at our data we found almost equally distributed labels close to 10% as seen in Figure-2. However we did not find any Null or duplicate values but the range of values are between 0 and 255 which can make the process of classification slow sometimes and algorithm like Logistic Regression may not even converge at less number of iterations. So we have dealt with the problem by changing the range of values by Normalizing the pixels between 0 and 1.

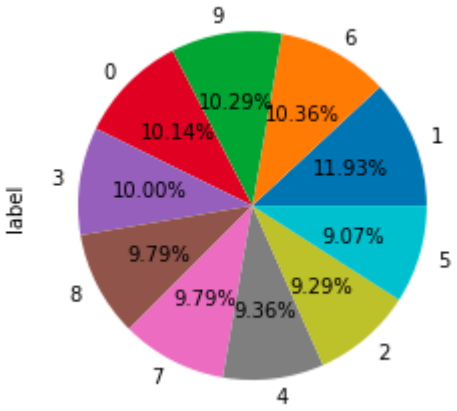

Figure 2: Distribution of labels in dataset

We have chosen Python [4] based libraries such as pandas [5] for data handling, matplotlib [6] for visualizations and scikit-learn [7] for modelling. All of the three mthod's implementations and comparisons are discussed below in the upcoming sections individually.

II. Naive Bayes

Naive Bayes uses the Bayes Theoram of conditional probability to find the best suitable classes. This sklearn based package calculates the class probabilities and uses it under the hood to find the probabilities for each data point for each class and assign the label to data point to the class with maximum probability. Data values in the range 0 and 1 are most suitable for this kind of problem. As we already have this from the normalized values, we do not need to perform any extra pre processing.

We have chosen BernauliNB from sklearn to perform this task .We had option for MultinomialNB and GausianNB but this is not the type of data suitable for Gaussian because the distribution is not Gaussian. MultinomialNB is best suited for binary values and after our initial run with both MultinomialNB and BernauliNB, we chose to go for BernauliNB as MultinomialNB is much faster than the BernauliNB but the latter gives better results as shown in Figure-3.

```
BernoulliNB has taken 0.016265392303466797 seconds and test scores are as follows:

Accuracy: 0.8107142857142857
Precision: 0.8105518433179724
Recall:0.8068667603150361
 ********************
MultinomialNB has taken 0.005975008010864258 seconds and test scores are as follows:

Accuracy: 0.8071428571428572
Precision: 0.8183951238158398
Recall:0.8014089996848618
```

Figure 3: Performance results of BernauliNB vs MultinomialNB

For hypertuning of parameters there are not so many options in BernauliNB so just after tuning 2 hyperparameters in the range '*alpha':[0.0001,0.001,0.1,0.25,0.5,0.9]*

and *'binarize':[None,0.0001,0.001,0.1,0.4,0.8,1.0]* we got *alpha*=0.001,*binarize*=0.4 producing the score metrices as:

Accuracy: 0.828
Precision: 0.830
Recall:0.825

III. K Nearest Neighbor (KNN)

KNN or K Nearest Neighbor is a technique based on finding the neighbors with similar set of attributes. Results of KNN is highly dependent on selection of number of neighbors where neighbor is defined as the closest data point to the candidate in terms of L-1 or L-2 distances. In a dataset with N points, neighbor=1 sepcifies that there should be only 1 data point which has to be considered and neighbor=N-1 specifies that every data point is considered as a neighbor and for the pat of getting results, both are bad techniques. We have selected default hyperparameters with n_neighbors=5 and distance metric L2 (Euclidean distance) and we have got a result as given in Figure-4:

```
KNeighborsClassifier has taken 0.060454368591308594 seconds and test scores are as follows:

Accuracy: 0.8714285714285714
Precision: 0.8763673263583852
Recall:0.866676139952002
```

Figure 4: KNN test scores with default hyperparameters

After Choosing our hyper parameters as ***'n_neighbors' = [1,3,5,7,10]*** & ***'p' = [1,2]*** we got best parameters as ***n_neighbors=3*** and ***p=2*** giving us the results as follows

Accuracy: 0.885
Precision: 0.891
Recall:0.880
Which are obviously better than the default one.

IV. Support Vector Classifier (SVC)

Support Vector Classifier or SVC are type of algorithms for classification whih tend to find the best fit line for 2 D data points or best fit hyperplane which seperates the data points clearly from one another. Here best fit is defined as the width of hyperplace which maximizes the ditance between classes. A data point is penalised (denoted by parameter C in SCV module) by the algorithm if it is found to be on the wrong side of the plane than to which it belong. Support vectors are those vectors or data points which are on the edge of the class boundry. We can define the harshness of penalty by C where larger the value of C, harsh will be the penalty for mis classified data points. With default settings of C as 1 with ‘rbf’ kernel and degree 3, we got the below results:

Accuracy: 0.946
Precision: 0.947
Recall:0.944

But after choosing the C,degree and kernal from ***‘C':[0.01,0.1,1,5,10], 'kernel':['linear','rbf']*** and ***'degree':[1,2,3]*** respectively we got to know that imposing a harsh penalty and decreasing the degree of freedom to 1 , we can have results as given n Figure-5 below.

```
SVC has taken 0.8659791946411133 seconds and test scores are as follows:

Accuracy: 0.9464285714285714
Precision: 0.9471202483222945
Recall:0.9453041530627736
 ********************
```

Figure 5: SVC results with hypertuning

Which slightely better than the default ones.

V. Algorithm performance comparison

Out of the 2 choices of SVM and KNN, SVM tends to produce the better results in every metric be it accuracy, precision or recall than KNN because SVM tends to deal better with outliers and it penalizes the data points if they are classified as wrong by the algorithm. On the other hand KNN tries to find the similarity between the data points and as we know what we have is image data, it makes it very ifficult to find the similarity between images as a single pixel can change the similarity between 2 vectors. It can be thought of as an example that {1,7,4} will show some similarity when depicted due to some common regions and {0,8,9} will also show some similarities which makes it very difficult to find a neghbor.

Training time of KNN is way less than that of SVM as we have just 1400 data point and it calculates 3 neighbors for each of the 1400 points within a domain of 784 attributes. On the other hand, SVM tend to find a place iteratively again and again until the change in loss from the previous iteration has reacher a threshold limit. It iteratively tends to find the “BEST” hyperplane which seperats the data points clearly. That is the reason why KNN is faster but SVM is producing better results. Figure-6 shos the test metrics scores of the two algorithms.

```
SVC has taken 0.8406383991241455 seconds and test scores are as follows:

Accuracy: 0.9464285714285714
Precision: 0.9477795037206802
Recall:0.9447638249362387
 ********************
KNeighborsClassifier has taken 0.06197834014892578 seconds and test scores are as follows:

Accuracy: 0.8714285714285714
Precision: 0.8763673263583852
Recall:0.866676139952002
 ********************
```

Figure 6: Performances of SVM vs KNN based on metrics and training time