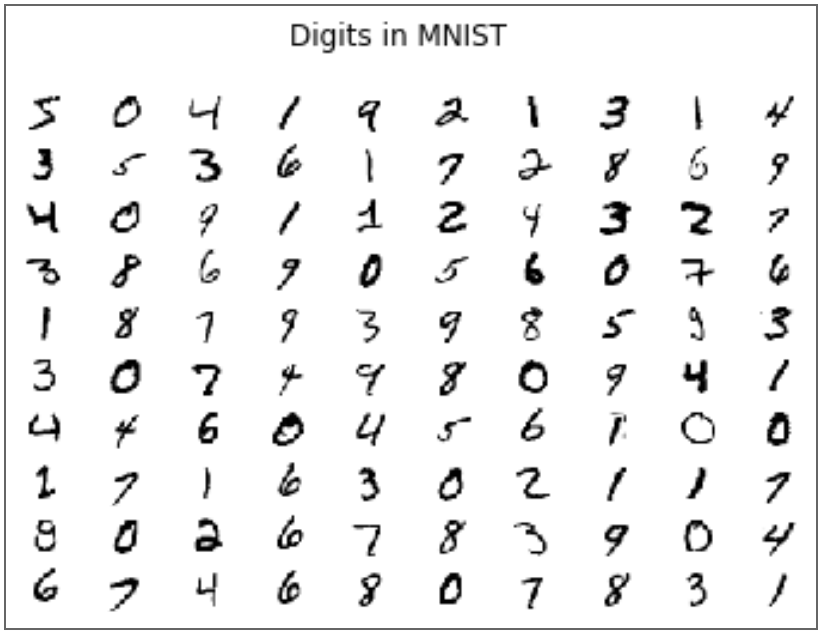
MNIST handwritten Image Classifier



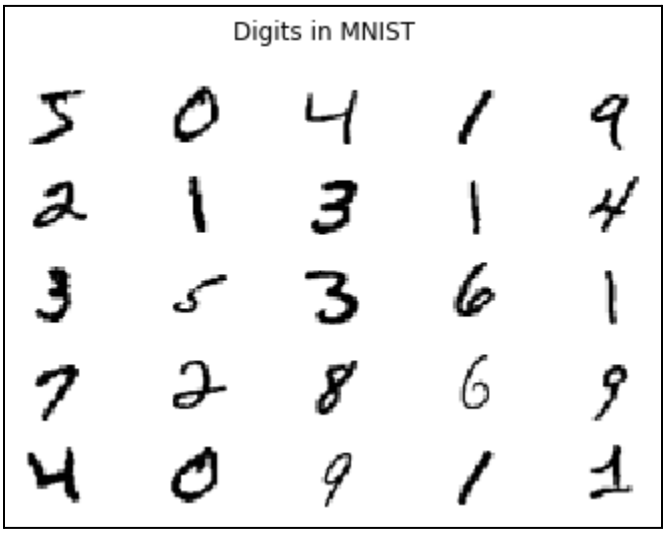
Shubham Kaushal
Contact: +91 7045 603 792
shubhamkaushal765@gmail.com
Gmail | LinkedIn | GitHub

Table of Contents

Dataset

Dataset downloaded from: https://datahub.io/machine-learning/mnist_784#resource-mnist_784_zip

MNIST 784 data is a collection of 70,000 labeled images of handwritten digits from 0 to 9. Each digit is represented by a 28x28 2D array (i.e., 784 columns), and it is divided into training and testing sets (60000 and 10000). This dataset is widely used for training and testing various image processing systems.



The task is classification. We have to predict which digit each image represents by matching it to its label. The present error rate (without deskewing/shiftable edges or using neural networks) is **2.8%*** using a single model. Here I have tried to build a stack of models, with each layer training on the previous layer's output.

*Note: The lowest error rate of 0.17% has been achieved by CNN using data augmentation.

Data Cleaning and Preprocessing

The dataset does not have any NaN or missing values. The images are represented by a 28 x 28 matrix where each place represents a pixel. The values represent the intensity of the pixels at that place. It ranges from 0 to 255, with 0 being white and 255 being black. We have scaled the data using StandardScaler(). It scales the data so that its mean is 0 and the standard deviation is 1.

Note: In this case, standard scaling the data may or may not work in the desired way. This is due to the fact that all the features of the MNIST dataset have the same scale (0 to 255) that has integer values. Scaling will bring it down to mean=0 and std=1, but it will then have floating-point values (prone to significant-digit errors.)

Models

ElasticNet

The ElasticNet model is a regularized linear model between the Ridge model and the Lasso model. Ridge model uses the “l2” regularization technique on a basic linear model, whereas the Lasso model uses the “l1” regularization technique. ElasticNet has “l1_ratio” as a hyperparameter to control the mix of ridge and lasso techniques.

SGDRegressor

Stochastic Gradient Descent (SGD) is an iterative method that tweaks parameters to minimize a cost function (gradients) at every step. It uses random instances at each training step and generally works well for large datasets.

KNeighborsClassifier

K-Neighbors is a simple, non-parametric classification method. An instance-based learning approach where a new instance is classified based on the plurality vote of its k-neighbors. As this model is based on distance calculations, the model’s performance largely depends on the scale of data.

RandomForestClassifier

Random Forest is an ensemble learning method that uses several decision trees to make decisions. For classification tasks, the output by the model is the class selected by most trees. Random Forest models are a popular choice as it requires very little preprocessing and provides reasonably good predictions.

AdaBoostClassifier

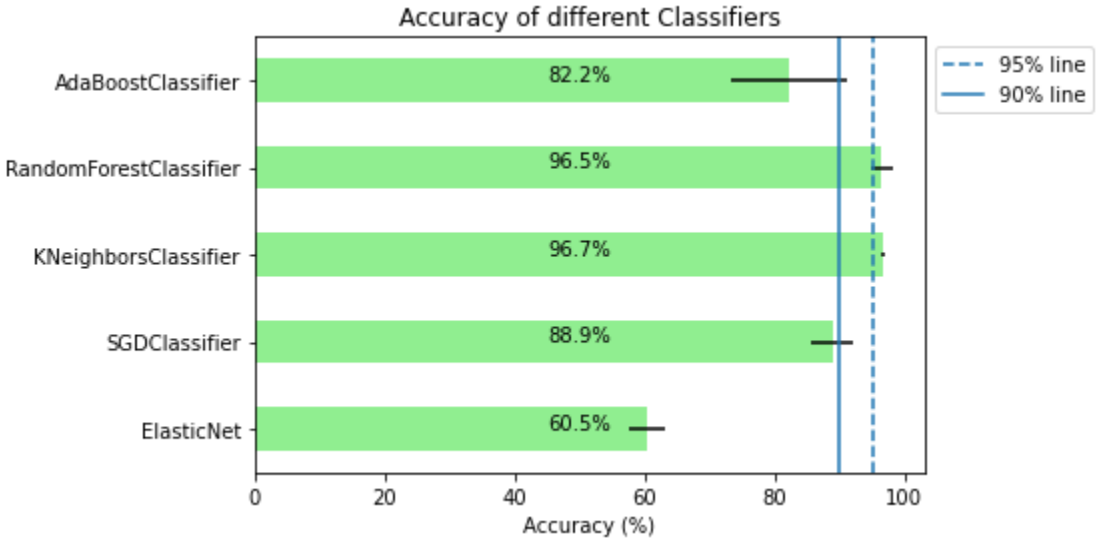
AdaBoost (Adaptive Boosting) is another ensemble learning method that uses a base classifier (here, DecisionTree) to make predictions. It sequentially improves the model by assigning more weights to the misclassified models in the next iteration. A second classifier is trained using the updated weights, and the process repeats. As this model works sequentially, hence it cannot be parallelized. That is, it scales poorly as the training set increases.

Comparing the performances

Cross-Validation

K-fold cross-validation is a great way to evaluate the models. Here I have used k as 3 (i.e., 3-fold cross-validation.) It randomly splits the data into three parts, keeping one part for validation, and then trains the model on the remaining two parts. Therefore we get three accuracy scores.

The plot below shows the mean of the accuracy scores, and the black line at the tip represents the standard deviation. (The standard deviation is scaled ten times before plotting for better insight)



The performances achieved by KNeighborsClassifier and RandomForestClassifier are above 95%. AdaBoostClassifier and SGDClassifier achieves 80% to 90% accuracy.
And the performance of ElasticNet is poor, with only 60% accuracy.

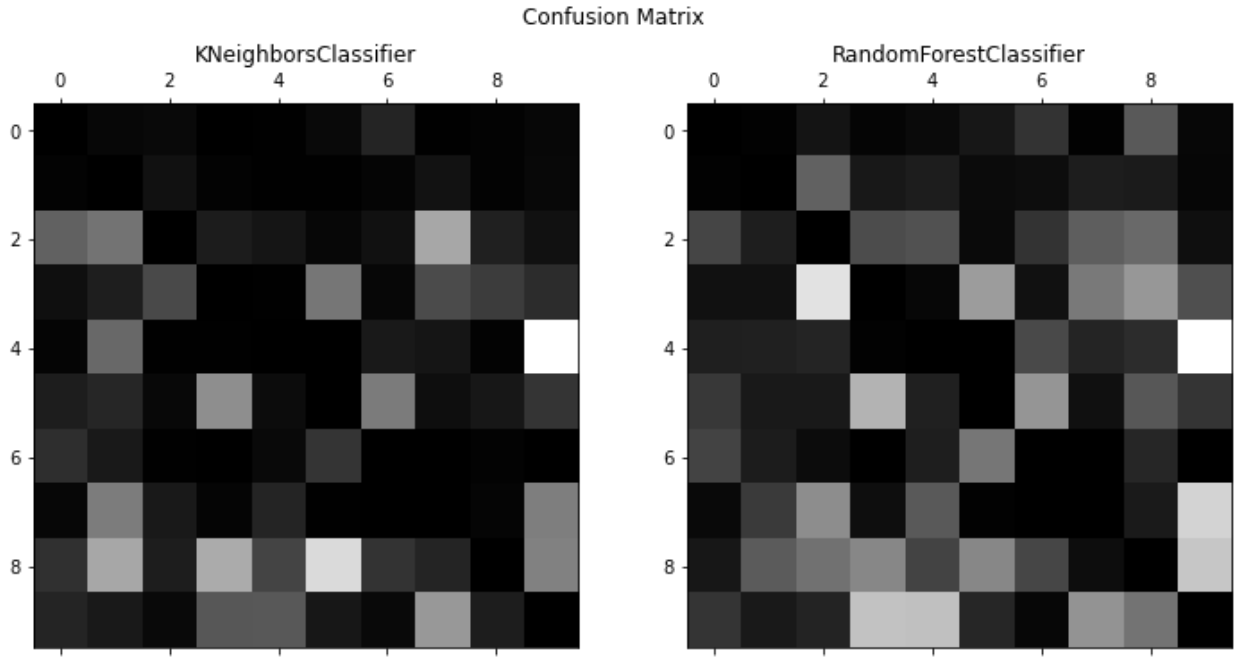
ElasticNet is a simple, regularized linear model and is severely underfitting the data. Therefore we need a more powerful model to make predictions.

KNeighbors and RandomForest are good predictors in this case. AdaBoost is a powerful model and is probably overfitting the training set. SGDClassifier is another powerful model, but it is unclear if it is overfitting or underfitting.

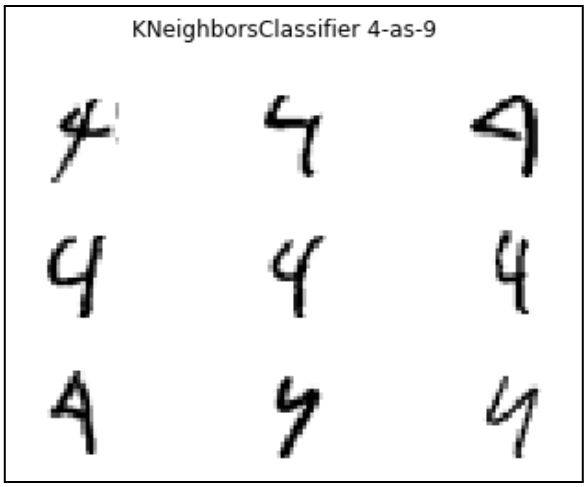
Misclassified values & Hyperparameter Tuning

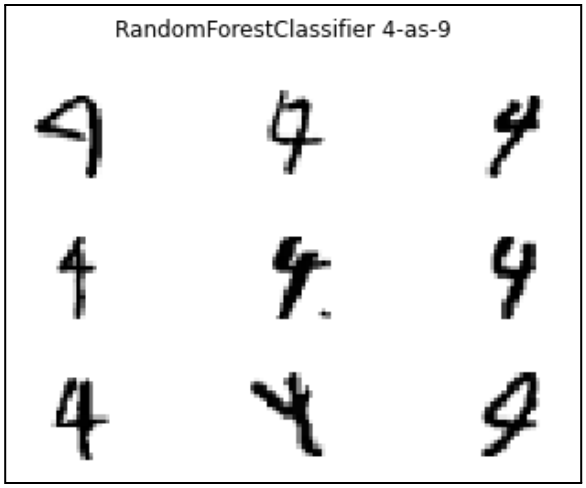
Confusion Matrix

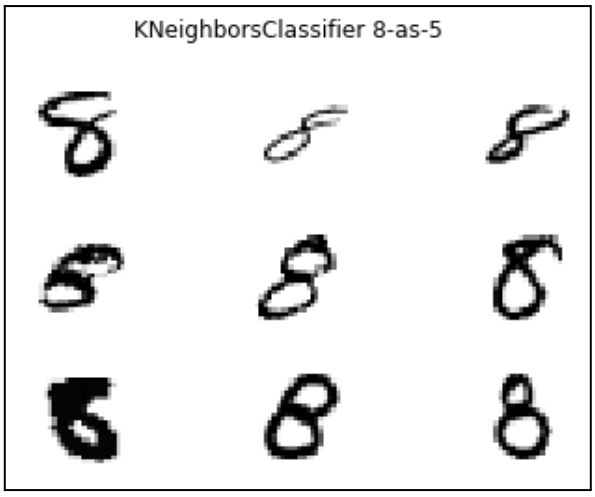
The confusion matrix is an excellent way to visualize any multilabel classification task. It allows us to see all the correctly classified and the misclassified instances.
After removing the correctly classified instances, the graphical representation of the confusion matrix is shown below. (KNeighbors and RandomForest)

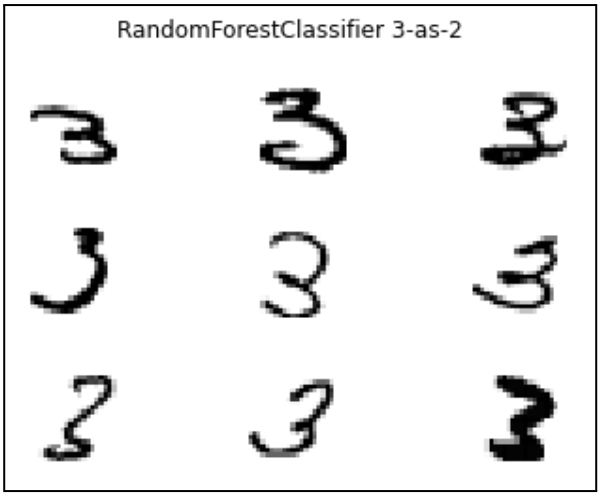


From the confusion matrix, we get a lot of information about the classification task. We see that the confusion matrix of RandomForest has a lot brighter patches than that of the KNN model. A lot of 4s are mislabeled as 9s in both models. Also, in RandomForest, a lot of 3s are misclassified as 2s. We can look at some of the 4s and 3s that are wrongly classified from their respective models.





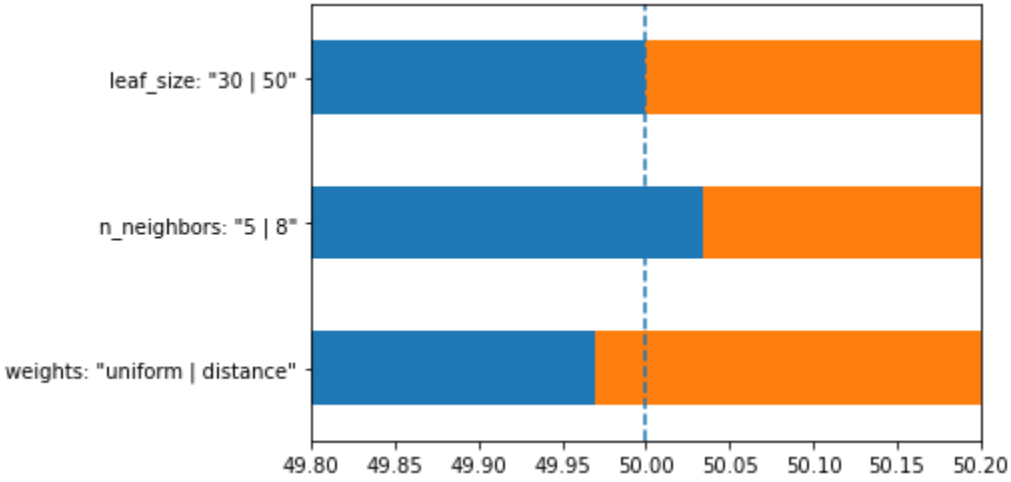




We see that some of the 4s really look like 9, as seen from the human eye. KNN performs poorly for 8s, and RandomForest performs poorly for 3s. We can build a better model to take care of these anomalies, but let’s move to hyperparameter tuning for the KNN algorithm as it provides maximum accuracy with small errors values.

Hyperparameter Tuning (GridSearchCV)

We trained various models with different values of the hyperparameters. Due to the constraints on the computational power, different parameters are tuned separately with only a couple of values. The results can be seen below:



We tune three hyperparameters: leaf_size, n_neighbors, and weights. Changing leaf_size does not affect the accuracy. The model performs better with fewer n_neighbors (=5) and weights set to “distance.”

Results with KNeighborsClassifier

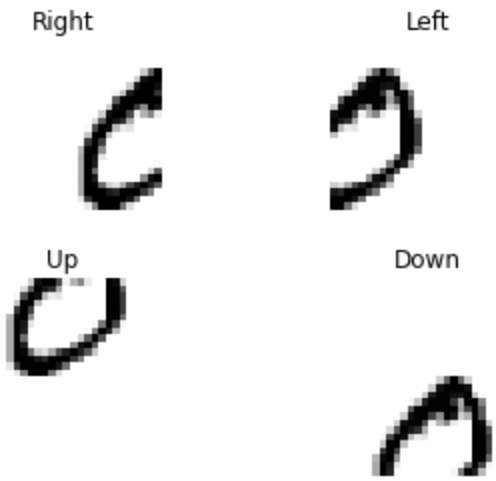
When using KneighborsClassifier, the model achieved **96.7%** accuracy for cross-validation. After hyperparameter tuning, the model was able to achieve **96.91%** accuracy.

Improving Performance

Data Augmentation

Data Augmentation is a technique in data analysis to artificially increase the amount of data by adding slightly modified copies of existing data. This is done by allowing small translation, rotation, cropping of the images. If applicable, changing lighting conditions of images (modifying RGB filters) also add to data. Data Augmentation acts as a regularizer and reduces the overfitting of data.

For the MNIST data, the images are translated 1 unit in all directions individually (up, down, right, and left) and added to the data set. The resulting dataset has five times more instances than the actual dataset, resulting in a better model. The below images are translated by ten units to show the effect.



The selected model, KNeighborsClassifier with the selected hyperparameters, is trained on the new dataset. This algorithm is computationally intensive for large training sets, so it might take a while to train. We can further speed up training by using dimensionality reduction techniques. The techniques are not used here as they usually lead to some information loss, reducing the model's accuracy.

Results afters Data Augmentation

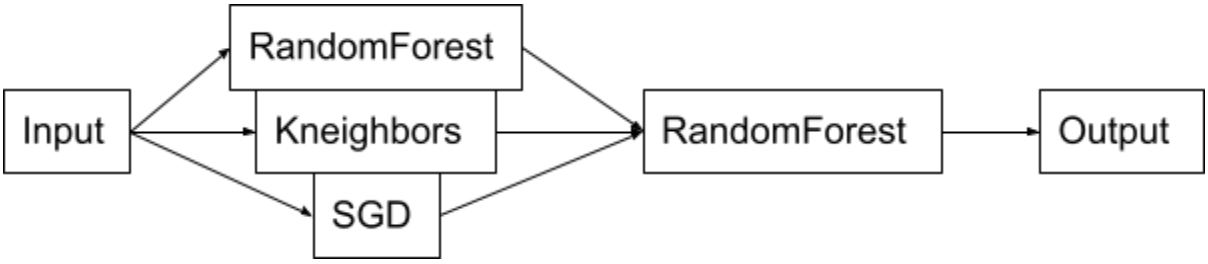
After data augmentation, the model was able to achieve **97.66%** accuracy.

Stacking Models (Custom Classifier)

We built a two-layer classifier using RandomForest, KNeighbors, and SGDClassifier in the first layer and RandomForest in the second layer (output). The idea is to use the second layer on the predictions on the first layer.

The first layer contains models that are independent of each other in their process of making predictions. RandomForest uses average results of all DecisionTrees, KNeighbors looks at instances nearby, and SGDClassifer is a gradient descent model that works by reducing cost function at every iteration. These models make independent predictions of the test data, and then the predictions are fed to the second layer, where another RandomForestClassifier is used to find the final prediction.

Model Diagram:



Results after Data Augmentation & Stacking

After using the model stack on the augmented training set, the model achieves **97.92%.**

What more?

Creating Complex Stack: More complex stacking models can be built to improve the accuracy further. But with limited computational resources, it may take a lot of time to train it.

Data Augmentation: More variation of the data can be augmented to the training set. Images with more translation, diagonal translation, and rotation can be used. Also, zooming in and zooming out the images may be added to improve the results.

Unsupervised Clustering: We can use unsupervised clustering as a preprocessing step to further improve our results. It groups similar instances into the provided number of clusters. If the correct number of clusters is used, It may significantly boost the supervised models' performance.

Neural Networks: Neural Networks are a very robust and flexible machine learning tool that can be used to make predictions. Depending on the structure of the neural network, it may have thousands or even millions of parameters. Deep Neural Networks (DNN) or Convolution Neural Networks (CNN) is well suited for this job. With, CNN error rates of 0.17% have been achieved on the MNIST dataset.

References

MNIST database
Elastic net regularization
Stochastic gradient descent
k-nearest neighbors algorithm
Random forest
AdaBoost
Confusion matrix
Data augmentation