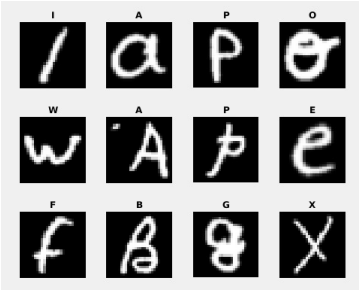
Data Preparation and Classification

In this project, we aim to accurately predict and classify images using the EMNIST dataset. Our goal is to train multiple machine learning models to correctly interpret the letters presented on screen in the EMNIST dataset through supervised image classification. The EMNIST dataset itself contains 26000 handwritten letters along with numerical labels indicating their position in the alphabet. Using these machine models, we are going to create training and testing subsets that split this dataset at random which are then used by these machine models so that they can learn to correctly predict the images in the dataset. We will collect the accuracy of these models and then compare with other built-in MATLAB functions to see how our models compare. By training these models, we can use them later on to accurately predict handwritten letters in other fields, they could be used to scan through exam papers that may be hard to read and output a neatly artificially written copy of students' answers. It can be applied to any similar situation where handwriting is involved, after rewriting out the text accurately it could even then be used with translation programs to quickly translate large pieces of text for those who can't read English.

Figure 1: Example Data from EMNIST



The dataset contains an array of 26000x784 double values which are compiled into the images above, a key string of the alphabet and a 26000x1 array of labels corresponding to the images. To split the data randomly I used the dividerand() function which allows you to define a size value, a test set value and a training set value. I used this to split the data 50/50 randomly mixing indexes into each. The key is applied to the labels so that we can see the alphabet representation instead of the numerical. The images in the array needed to be resized as 28x28 double arrays so that they could be presented individually.

To train the models and solve this problem I have used the KNN(K-Nearest-Neighbour) algorithm which takes a point in a set of data and takes its K nearest neighbours and predicts the value of the current data. I also used the built-in fitcknn function as this is also a KNN algorithm which gives a good base to compare our own trained models too. I have also used the fitcecoc function which produces an Error-Correcting Output Codes model to compare how efficient our KNN is compared to other types of machine learning algorithms. To train the models I passed in the sets I created above through the dividerand function and then calculated the accuracy, time and correctness of each for evaluation. For the KNN model I designed I used a replicating matrix to check each testing image against the entire set of training images for every testing image. I ran each 3 times and calculated the mode of the values to get ensure consistency.

MyModel: L1: Times: 229.56s, 228.79s, 230.12s Mean: 229.49s Accuracy: 0.7734, 0.7862, 0.7799 Mean: 0.7798 Correct Predictions: 10054, 10220, 10139 Mean: 10137. L2^2: Times: 241.72s, 237.62s, 239.80s Mean: 239.71s Accuracy: 0.7518, 0.7524, 0.7578 Mean: 0.754 Correct Predictions: 9774, 9781, 9851 Mean: 9802. The model that I configured myself seems to have a near 80% accuracy rate and a consistent number of correct predictions. Since it is an unoptimized algorithm it takes a considerably longer amount of time than the other 2 models.
Fitcknn: Times: 14.53s, 14.63s, 15.36s Mean: 14.84s Accuracy: 0.7766, 0.7818, 0.7833 Mean: 0.78 Correct Predictions: 10096, 10163, 10183 Mean: 10147. The in-built KNN model seems to have a similar accuracy and correct prediction count as the function above. The results are interchangeable as in some cases it beat it and in others it was slightly lower. The timing however is much faster than the unoptimized algorithm taking a fraction of the time.
Fitctree: Times: 1.32s, 1.20s, 1.38s Mean: 1.3s Accuracy: 0.5585, 0.5637,0.5581 Mean: 0.5601 Correct Predictions: 7260, 7328, 7255 Mean: 7281. The in-built tree algorithm is the fastest out of the three I chose for this project taking no longer than 1.4 seconds consistently. The accuracy and correct predictions however are much lower than the previous. A quick but inaccurate solution.

In conclusion, I think there is a clear choice when coming to solve this problem. The in-built KNN function may have a slightly lower overall accuracy, but the difference is neglible and the time advantage more than makes up for it taking on average just below 5% of the time the unoptimized function takes. The decision tree may be quicker but a problem like this requires accuracy, or the solution is almost useless for practical use. The time is still quite low relatively and could be used to scan through documents in a convenient amount of time. In future, optimising the time even further and perhaps trying to predict the letters more accurately could provide huge benefits to this program, but it is satisfactory as it is for problems like scanning exams and smaller text documents.