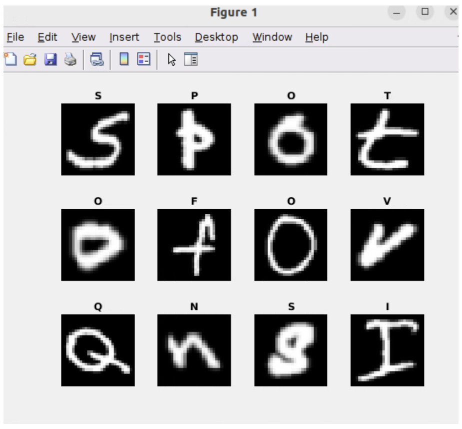
A look into using KNN and Decision Tree model to identify handwritten characters.



1. **Introduction:** Using the EMNIST dataset to classify hand drawn letters and assess different machine learning models in their ability to classify handwritten letters. This could be potentially useful in handwritten documents to understand what letters have been written, without need for a human.
2. **Data and Preparation**: The dataset is held in a variable called dataset from the "dataset-letters.mat" where it contains arrays of images, key and labels, images are stored as a 26000x784 where 26000 is number of images. We used randperm() to extract 50% of the data for training and other for testing, same for labels, so we can train the model and test it. Here is a 3x4 grid of images generated by 12 random numbers from 1 to 13000 because I used the training set. I selected 12 images at random and reshaped them into 28x28, then generated a subplot with these; the labels were in digits, so I converted this to a character by using the key variable in the dataset. This show what our data is looking like and what labels are associated with them, I can clearly see that these are correct, so I know that the labels and features are working correctly.
3. **Methodology:** I implemented the KNN and chose K as 26 since there are 26 letters in alphabet. Then for each testing feature, we calculate the distance from all other points already mapped and find the shortest point which we then class that point as being that label which it is closest to - we do this by using the Euclidean distance, another distance measure will be used such as Manhattan. I chose the Manhattan model because it was like the Euclidean version, so I wanted to compare its times to see if it was quicker.
   We repeat this for all testing features while recording time taken, and finally find the amount of correction predictions to get the accuracy of the model. The dataset size had been varied from 2000,5000 to 13000 for testing features, to provide variation in the data so different data sizes can be compared to see how the model scales. I also compared this with the built-in KNN model and a decision tree. The accuracy and times were calculated, this allowed me to see how close my own model is to the tested version which is known to be working correctly. I had chosen a decision tree since it was the other model, I was familiar with and wanted to compare it to the KNN.
4. **Results:** The observation here is that from using a 50% split, as we use more training data, the accuracy of the values increases, however the time taken to train the model gets significantly longer to train, without a big jump in accuracy after around 13000 training images.
   Compared to the Built in KNN model, my times were significantly slower, however the model only had slightly higher end accuracy when using the 13000 images, so the overall accuracy had not been that different, meaning my model had been near perfect to the tested version.
   The decision tree had the lowest times; however, the accuracy was only around half, which meant it was not as useful as compared to a KNN model when it came to identifying hand drawn characters.
   I had also used Manhattan distance to compute the KNN model, however this gave lower accuracy in the end but takes around same time to compute, hence giving worse performance than using Euclidean distance, so it is not as useful.

Dataset_Size	Accuracy(%)	Time(s)
2000	56.25%	7.607
5000	65.42%	55.406
13000	73.31%	857.578

Dataset_Size	Accuracy(%)	Time(s)	//BuiltIn Knn
2000	64.55%	1.882	
5000	71.9	10.466	
13000	78.04	83.667	

Dataset_Size	Accuracy(%)	Time(s)	//decision tree
2000	43.1	0.767	
5000	48.08	2.32	
13000	56.9	7.397	

Dataset_size	Accuracy(	Time(s)	//manhattan distance
2000	54	7.461	
5000	64.3	53.847	
13000	71.35	849.223	

5. **Conclusion:** In this showcase, the two models compared were the KNN and Decision tree. The table showcases the results and while the decision tree computes much faster, the KNN model is more than 20% more accurate in its evaluation of the model, which makes up for the extra time. Overall, the KNN model is more useful as it gives higher accuracy in the end which is more valuable than timing in this case; since it is a program for identifying text, it can be assumed that it is not critical to have quick computation times, so in the end the KNN model would be more useful, but this would depend on the context.