Automated Recognition of Letters and Words with AI—Joshua George

Introduction To Project

Objective: The objective of the Project is to develop a character recognition program that would automatically identify text present in images and convert these into a machine-readable character array.

Background: Computers understand input from Keyboard and Mice, and generally are not able to read printed literature or handwritten notes. Traditionally Data Entry Operators used to enter this information into a Computer. This program would allow computers to scan and read printed books, magazines, filled forms, handwritten letters, etc. thereby eliminating the need of a Data Entry Operator to manually input these into a Computer. This will speed up digitization of vast amounts of information that exists, and new ones that get created daily.

Literature Review: A vast variety of literature is available online that highlights numerous different approaches to character recognition. The Python Deep Learning API [1] and Machine Learning In Python [2] provide valuable information about character recognition approaches. Libraries such as SkLearn can be used to create models.

Printed Text Recognition

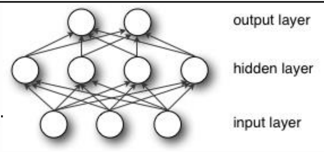
Analysis: Initially, when starting this part, it showed there was the need for individual letters to be extracted from both the training and testing files that had been provided so that the classifier could be trained on what the sample letter would look like. These letters would need to be formatted to be of the same size to make it easier to extract features. In addition, I realized that there was the need to train both languages independently of each other in order to ensure that the classifier could recognize the letters more accurately. I then investigated the different models available on Keras[1] and SkLearn[2]. I then decided to use SkLearn.

Technical Issues: Some of the technical challenges that I had faced were a result of my feature extraction algorithm. It was unable to detect punctuation since my training set did not contain any. This also meant that it was unable to detect spaces, which meant that it would consider the whole text as one word. Since I did not implement word segmentation as my extraction algorithm would create rectangles around each letter, it led to a reduction in my accuracy since the output would be compared against spaces. Other issues I faced were setting up the libraries and running my code again with minor changes to the model would take a long time(over a few hours) as I was running the code on a Virtual Machine that was not able to perform all the calculations between layers as quickly as possible.

Processing Issues(data prep and feature extraction) And effect on **performance**: Initially when processing the images, I found that the features that had been extracted were of different sizes and would make it harder to train as some images may be misrepresented. In order to normalize the images, I had to resize them all into a fixed size. Since there were multiple letters in a single image, I extracted them individually by looking at smaller boxes which increased the run time.

Approach: When starting this part, there was an option to choose between keras and Sklearn. Due to incompatibility of keras with my VM, I decided to use SkLearn to create my models. I had decided to train individual languages separately to prevent labels from the second language that were similar from being misclassified. I resized all my images to ensure that all the letters were of the same size so that the features would be similar throughout my dataset and prevent errors from not being able to detect features. I initially created a Neural Network however it would classify all objects as the same class, so I decided to use other models such as a KNN(K Nearest Neighbours), SVM (Support Vector Machine) and RFC (Random Forest Classifier). I then tried a multilayer perceptron classifier (MLP) that was only able to identify objects as one class. I also attempted to create a Convoluted Neural Network(CNN), however there were issues with my input By creating multiple models for this part, I was able to contrast all different types of models and observe which model would be most efficient. In order to train the data. I had stored the training images in an array with another array to store the corresponding labels. Then, I would create instance of the models and fit the data. I then performed cross validation using SkLearn to choose random subsets and test model accuracies to help identify model best suited model for the task of recognising printed text.

Models



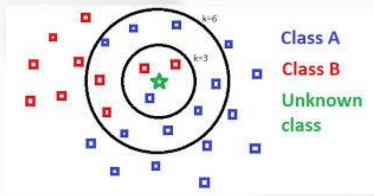
MLP: A multilayer perceptron (MLP) is a class of neural network (ANN).

An MLP consists of at least 3 layers of nodes: an input layer, a hidden layer and an output layer. Apart from input nodes, each node is a neuron using nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron as it can distinguish not linearly separable data.

Advantage: The main advantage is the fact that it can adapt quite easily during the learning stage. It also does not make any assumptions about probability density during training, thus making it less susceptible to bias.

Weakness: Due to the amount of backpropagation needed, it will take a longer time during the training stage.

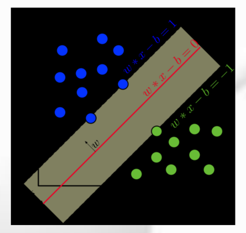
KNN



In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being clustered to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Advantage: It is highly effective, simple and easy to set parameters

Weakness: It is computationally expensive and each new item needs to be compared to all of the current classes, which means that the training time for the algorithm is high. It also needs the number of expected classes to be known in advance.

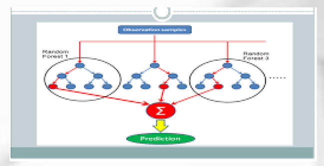


SVM

A Support Vector Machine(SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

Advantage: It is highly accurate, even if there are multiple different features to be identified.

Weakness: It can't be used for unlabelled data

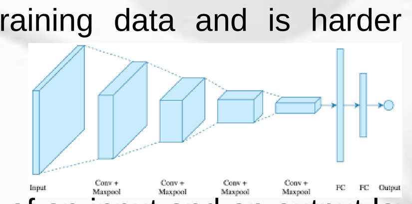


RFC

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

Advantage: It is highly accurate and training is performed in a short time, even if there are multiple different features to be identified.

Weakness: It normally overfits the training data and is harder to interpret.



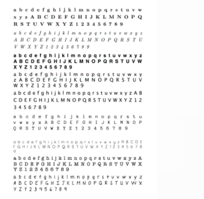
CNN

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution

Advantage: It has a high processing speed, with sub sampling to smooth the data.

Disadvantage: It can't learn temporal dependence.

Implementation: In order to attain the objective, I have created various classifiers –



- neural network classifier,
- random forest classifier,
- multi-layer perceptron classifier and
- support vector model classifier

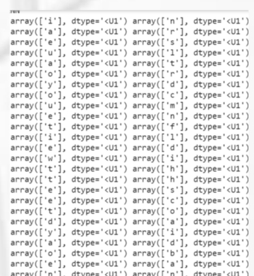
I used these classifiers to train the AI models using datasets such as the one shown above to recognize letters that have been extracted from texts in different formats, such as printed text in multiple languages such as English and Greek, handwritten letters and signatures to verify and classify them accordingly. When conducting research at the start of the project, I had found libraries to use such as SkLearn that would be used to create models and validation was performed to ensure that the expected text would be recognized correctly by the system.

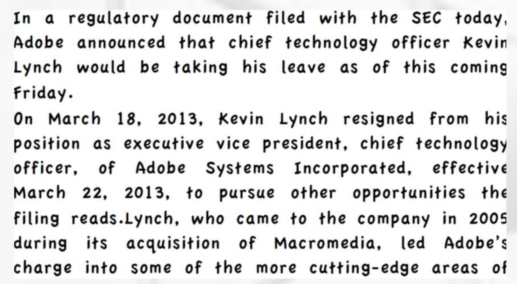
Initially before the training phase, I had to gather all the data from the files. In order to do so, I looped through all the training file, read each image and extracted each letter by creating rectangles the same size as a character. I resized all characters to the same size, then flattened the array of the character, and stored the data as shown above in trainingSet, with the label of the character stored correspondingly in trainingTarget.

Dataset: The dataset I am using is the OCR folder provided, with the training folder for training and testing folder for validation. I then created all the different training network algorithms such as MLP, KNN, SVM and RFC. I then fit the data so that it would loop through the trainingSet which had about 3300 elements(35 of each letter in upper and lower case as well as 0-9) and create a prediction using all training data for each model. I then used cross validation to check the accuracy. I also used SkLearn to get different analytics to evaluate my model. Then, I looped through the 3 Test Files which had over a 1000 elements for testing purposes and created predictions using each model for all the characters. I stored data for each model in an independent array and then did a visual inspection against the actual contents of the file for verification. As my initial models did not give me a good accuracy, I iteratively tweaked the hyperparameters for my models in order to help fit the training data a bit better.

Model	Cross Val1	Cross Val2	Cross Val3
MLP	0.90163934	0.92349727	0.83825137
KNN	0.89945355	0.93989071	0.85245902
SVM	0.9147541	0.94754098	0.87759563
RFC	0.91038251	0.93551913	0.86338798

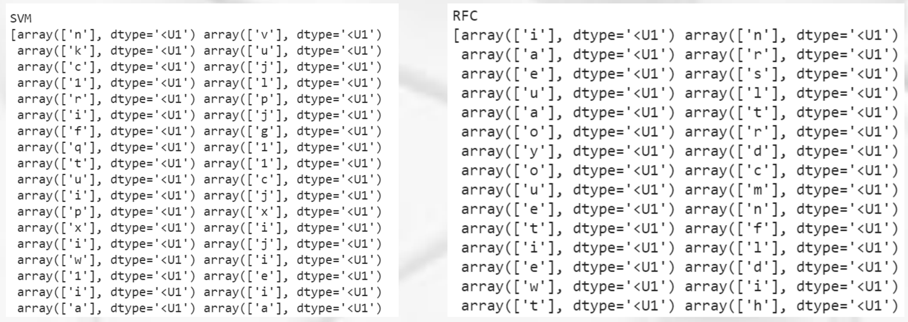
The figure on the left represents the output for the training of the KNN model, while the right side shows the actual file's contents. It is very similar, as indicated by a strong cross validation score. I used 5 nearest neighbours which is the default to avoid over/underfitting data.





The MLP model also had a reasonable accuracy cross validation, which is reflected in the output that was like the test file.

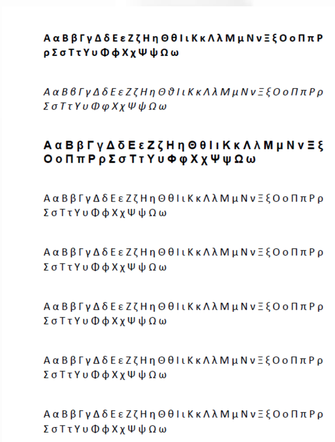
The SVM output suggests that it may have overfit to the training data and thus led to a low test data accuracy on inspection against file.



The RFC data is also accurate, as evidenced by the high cross validation score and match against the test dataset.

I then repeated the same steps for my second language, which I chose as Greek. I created my own data set for Greek by pasting letters in different fonts into MS-Word. I would train using the above steps for all the models and then I would create a test set by taking a third file with all the images. I created the model using new instances of the same classifier models.

As can be seen, some of my classifiers worked for Greek letters during cross validation. This was since the dataset was small so it was unable to learn features as efficiently, underfitting the data whilst others may have overfit. As seen for the Greek data, it was fine upon inspection of data.

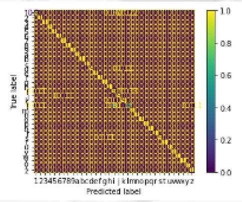


There was a large variation in cross validation as seen.

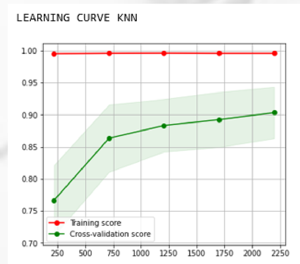
Model	Cross Val1	Cross Val2	Cross Val3
MLP	0.83939394	0.52727273	0.86322188
KNN	0.74363636	0.52459016	0.41347905
SVM	0.29090909	0.21857923	0.20765027
RFC	0.88181818	0.55009107	0.46994536

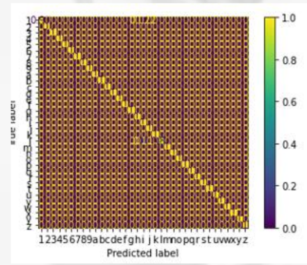
Evaluation: In order to evaluate the various models, I used SkLearn's inbuilt functionality such as cross validation, learning curves mentioned in the Implementation section. Since my models were used to classify object into multiple different classes(more than 2), I was unable to use an ROC curve to model my data. However, I used a learning vs error curve, confusion matrix for each model as well as classification report to compare different models for both languages.

MLP – It was unable to create a graph for English letters due to the large dataset. The confusion matrix shows that there were some letters that were consistently confused for others.



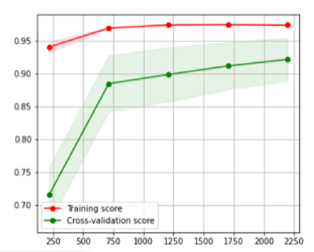
KNN

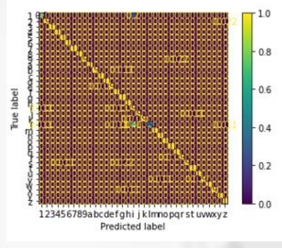




As seen the learning rate improves exponentially then flattens out as it is no longer able to learn as much between iterations which shows that it eventually becomes 80% accurate after all the iterations. It is generally not missing much as seen in the confusion matrix.

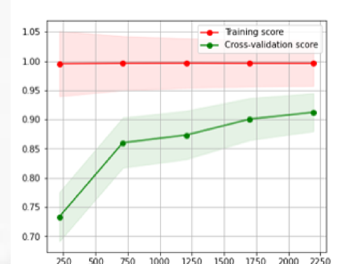
SVM

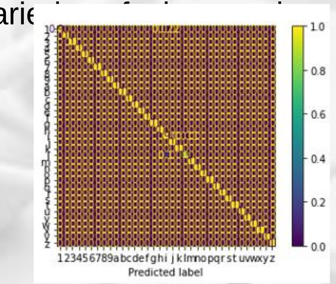




The SVM performs slightly better at the start whilst learning, however it ends up having a slightly more varied confusion matrix.

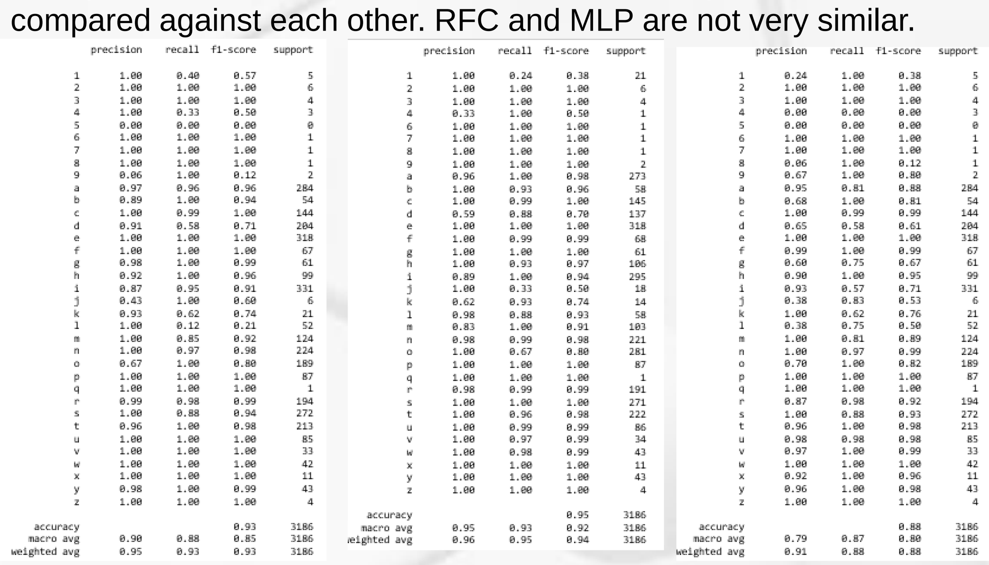
RFC





The RFC is better at classifying, as seen by the better initial learning rate and the less varied confusion matrix.

The pictures below show the classification report for all models compared against each other. RFC and MLP are not very similar.

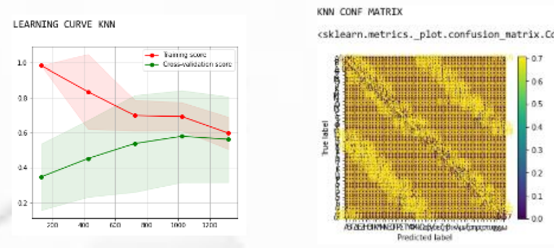


RFC VS SVM KNN VS RFC RFC VS MLP

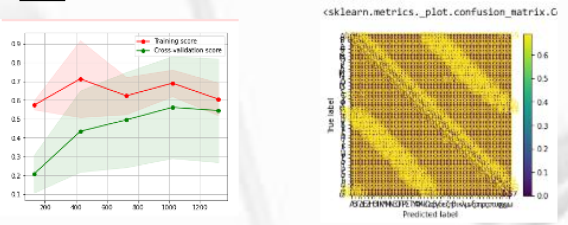
Greek

I created my own data set for Greek by pasting letters in different fonts into MS-Word.

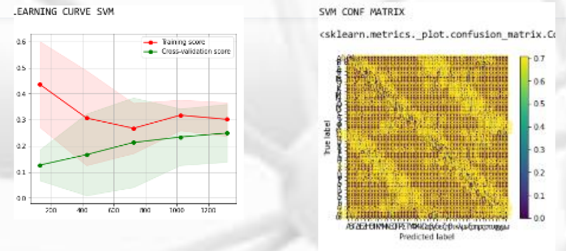
KNN



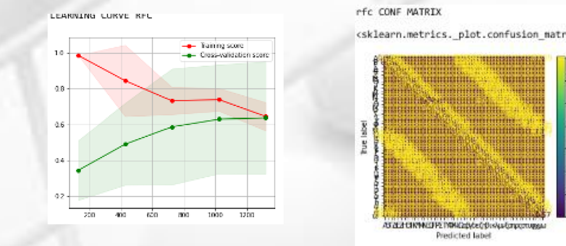
NN



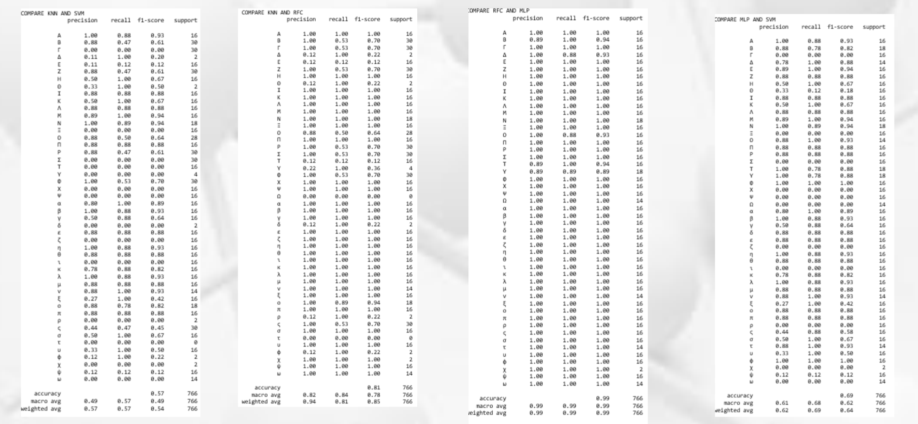
SVM



RFC



These graphs show the models were unable to fit the data as well, while the classification report suggest that models generally agree amongst themselves on the data's label, but only RFC and MLP have a very similar score.



Discussions And Findings

Overall, by analysing all my results, it would show that a MLP is reasonably suited for such a Task, provided there are enough neurons in every layer and a sufficient amount of samples. A SVM could be used for English based texts but not for Greek texts, which may be a result of not having a large enough dataset. A KNN would be useful if needing to model multiple different languages.

Recommendations

Based on my findings, I would recommend utilising an SVM if there is a small dataset as it can learn quickly. If there is sufficient data, I would utilise a RFC as it has the highest cross validation score

References:

1) Team, K., 2020. *Keras: The Python Deep Learning API*. [online] Keras.io. Available at: <https://keras.io/> [Accessed 7 June 2020].
2) Scikit-learn.org. 2020. *Scikit-Learn: Machine Learning In Python – Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/> [Accessed 7 June 2020].

Automated Recognition of Letters and Words with AI—Joshua George

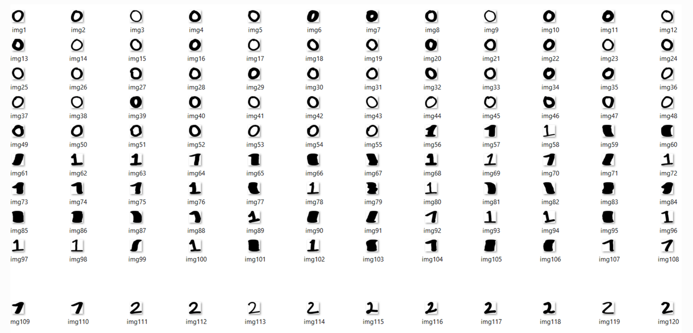
Handwriting

Analysis:

I realised that in order to create an effective handwriting recognition system, I would need to have a significantly large dataset in order to create a training and testing set. I therefore looked for datasets that had a wide range of different classes as well as a lot of examples per class, to make it easier to identify more similar letters.

Technical Issues: There were not many handwriting datasets that were open source available when I was initially looking. This then meant that a significant amount of time was spent trying to find a suitable dataset. Since I was also doing this task on my VM, it would take a lot longer to train the models. Due to deprecation of a few features, I had to modify my approach to include workarounds for the APIs that I was utilising.

Processing Issues(data prep and feature extraction) And effect on performance: I realized that the data was stored in individual folders for each label. So I had to manually loop through all the files and create a single folder approach so that I could label all the files more appropriately. This increased the run time significantly.



Approach:

I decided to use the same models that I had created for the previous task with some minor tweaking to accommodate the format of the new data. I realized that I would have to adjust the hyperparameters in order to fit the data. Since there were sufficient samples of every single letter(56 samples of 61 letters), I decided to put every 5th letter from my dataset into only my testingSet.

Implementation:

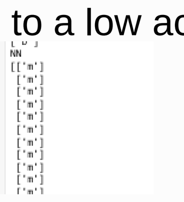
I looped through the files and used imread to read all the images. In order to convert the 2d array of pixel values into a 1d array that could be used for training, I used the ravel function in numpy. I then created my trainingSet and label by adding 4 out of every 5 images to it, whilst the 5th image would be stored in my testingSet. This ensured that the test Set was unique from the training Set.

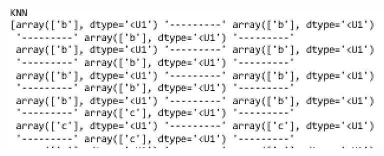
I then created my models by using the same library(SkLearn) and then iteratively tweaked my parameters to fit the model. Again, I used cross validation of 3 folds to test the accuracy.

Model	Cross Val1	Cross Val2	Cross Val3
MLP	0.03363636	0.03272727	0.03272727
KNN	0.70909091	0.78727273	0.70181818
SVM	0.76	0.79818182	0.72636364
RFC	0.71727273	0.76363636	0.67454545

I then predicted my test dataset for each model and stored the results in a separate array. I ensured that there was a total of 11 elements(one fifth of dataset) of each type being returned. Then, I created graphs for the learning rate and plotted the confusion matrix to test which letters were being mislabeled.

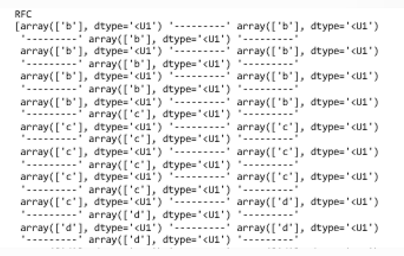
The neural network failed with multiple classes again and would assign everything to the same class again. This led to a low accuracy as seen in the cross validation score.





The KNN was able to classify the different classes appropriately as seen in the image. However, it was not as accurate as the first task which is seen by the lower cross validation score. This is due to a slightly larger variance in the images due to difference in the individuals who wrote the dataset, whilst pictures for the letter “a” and “o” being similar.

The SVM was not as accurate and may have overfit the training data again, since it mislabeled more of the testing data whilst having the highest cross validation score.

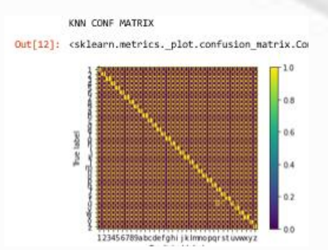


As seen in the image, the RFC was able to classify the training data reasonably accurately.

Evaluation:

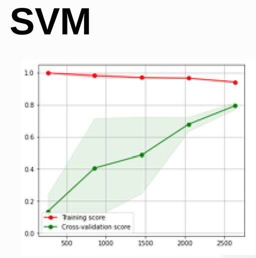
In order to evaluate my results, I again created graphs for learning score, cross validation score as well as a confusion matrix. Due to the amount of time taken to run models, I did not implement a classification report. Since there were still more than 2 classes, I was unable to create a ROC curve.

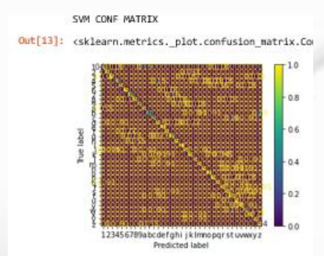
KNN



This shows that it progressively learnt how to do the tasks and was able to attain an accuracy of almost 80%, which was probably a result of noise in the data. It did not confuse many elements.

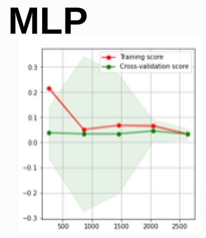
SVM

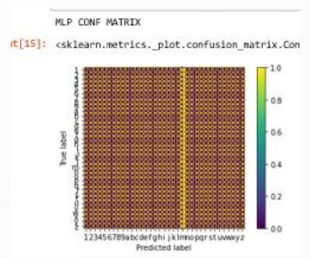




This shows that although it was performing well on the validation, it still hadn’t properly fit the data, which can be seen as there was a wide range of elements being confused.

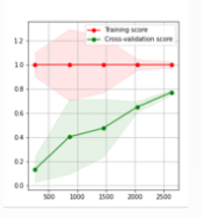
MLP

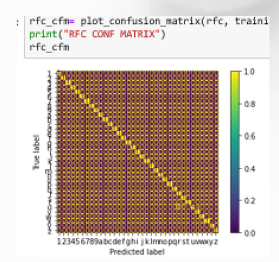




This again highlighted the fact that an MLP network is not very good at fitting data with multiple classes, and that it then predicted everything to be “m”.

RFC





The RFC was able to classify most of the data accurately, which can be seen by the improvement in the learning rate through time. It also had the minimal amount of elements being confused with each other.

Discussion And Findings:

Overall, RFC had a very good run and was only hindered by the fact that the dataset had images for objects in different labels that were extremely similar, such as “a” or “o”.

Recommendations:

I would recommend using an RFC for this task since the confusion matrix showed that there is minimal elements being mistaken for other classes, which would mean that there would be lesser erroneous data when trying to predict handwritten letters that are not part of the trainingSet.

Signatures

Analysis: I realised that I would only need to make minimal changes to my handwriting script to run the signatures. I realised it would be easier to train a classifier to recognise the whole signature rather than letters in the signature.

Technical Issues: N/A

Processing Issues(data prep and feature extraction) And effect on **performance**

A few classes of data were missing from my dataset. This meant that I had to manually look through the training folder to identify missing labels and images and avoid using those classes.

Approach:

I decided to use the same models that I had created for the previous task with some minor tweaking to accommodate the format of the new data. I realized that I would have to adjust the hyperparameters in order to fit the data. Since there were sufficient samples of every single signature(12 samples of 30 classes of signatures), I decided to put every 4th letter from each label into only my testingSet.

Implementation

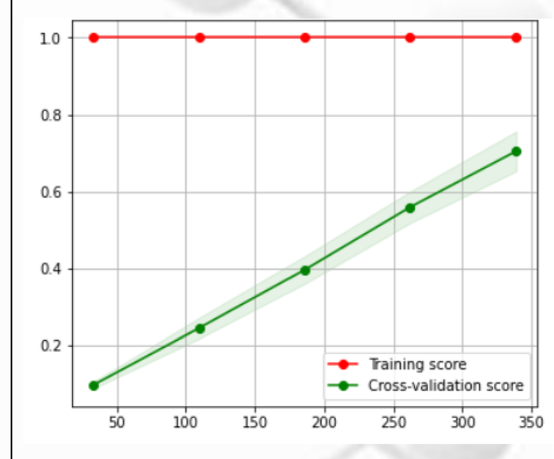
I looped through the training directory in order to get the training data, resized them to make it easier to identify features, then stored the array and its label. I then fit this data using an MLP, KNN, SVM and RFC. I then looped through my testingSet and would predict using each model and stored the value. I then performed cross validation and then displayed the learning curve and confusion matrix for each model. I inspected the output to see if it was classifying them correctly and adjusted the hyperparameters slightly to help fit the model.

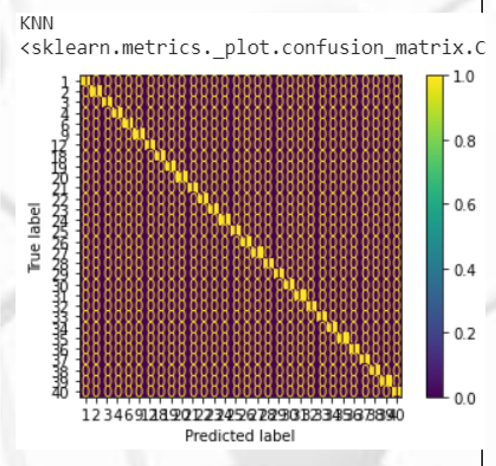
Model	Cross Val1	Cross Val2	Cross Val3
MLP	0.02272727	0.03409091	0.02272727
KNN	0.7875	0.7875	0.7375
SVM	0.71830986	0.78014184	0.77304965
RFC	0.8028169	0.87234043	0.84397163

Evaluation:

In order to evaluate my results, I again created graphs for learning score, cross validation score as well as a confusion matrix. Due to the amount of time taken to run models, I did not implement a classification report. Since there were still more than 2 classes, I was unable to create a ROC curve.

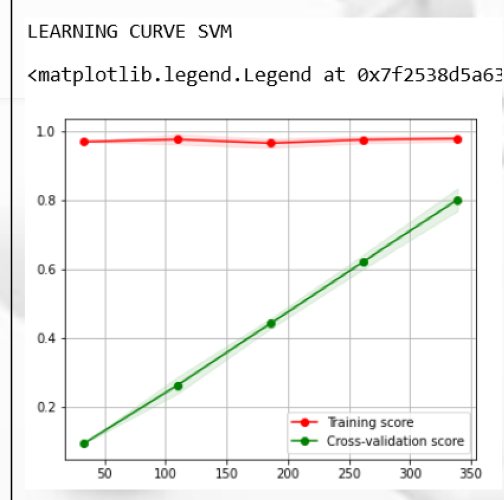
KNN

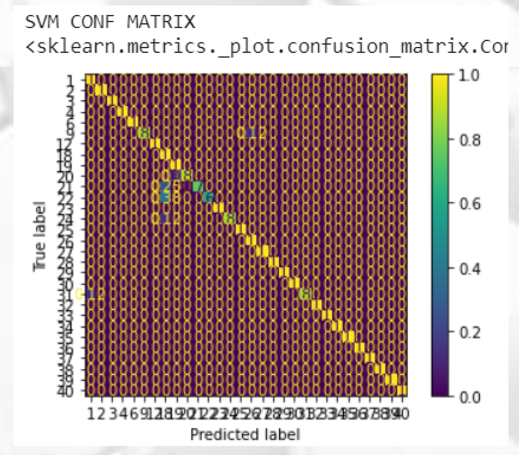




KNN is not as effective at this task as it was at other tasks, seen by low accuracy and slight confusion.

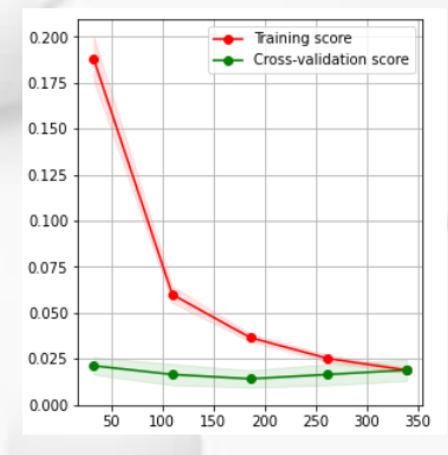
SVM

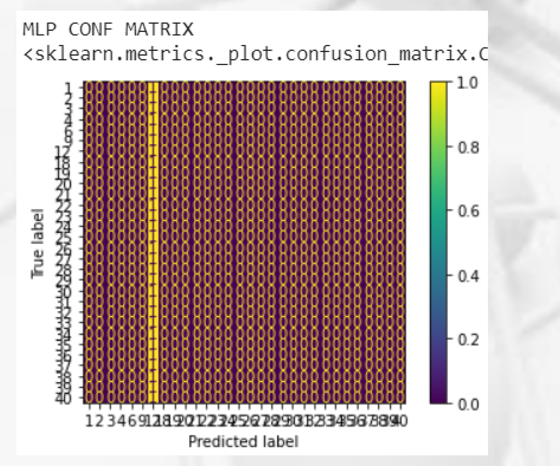




SVM would be able to categorise new data easily but suffered from confusion as witnessed.

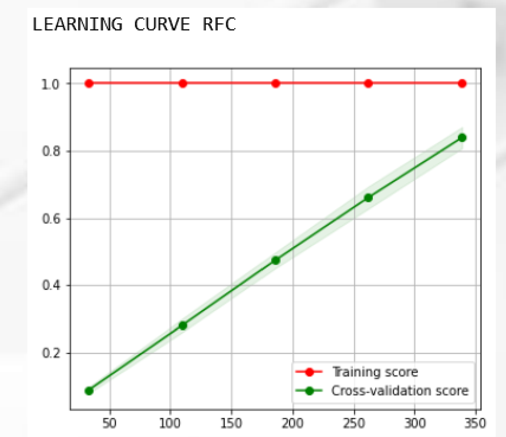
MLP

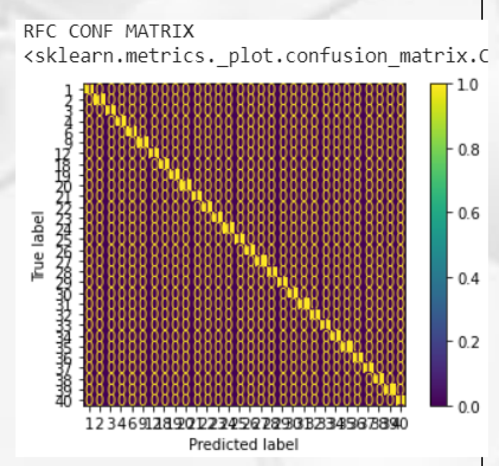




An MLP was unable to distinguish between the numerous different classes, which could likely be a result of overfitting the data.

RFC





An RFC had the highest accuracy and also low confusion.

Discussion And Findings

Overall, by analysing all my results, it would show that a MLP is not suited for such a Task. This is because there are not enough neurons in every layer for it to be able to clearly distinguish between these massive amounts of classes. An RFC would be more of an ideal choice as it is designed to find the most suitable class for data that is not entirely distinguishable.

Recommendations

Overall, I would recommend using either the SVM or RFC to predict as they would be able to create an accurate representation and are unlikely to get confused as much.