Team 34
Classification of Handwritten Numbers
Final Report

Baris Dingil, Isaac Joshi, Rishvanjay Maheshwari, Ali Yalcin

Introduction

Text and image recognition is an important technology that is used extensively in data conversion, natural language processing and security/surveillance. The goal of this project is the generate a system of linear classifiers, that given an image of a handwritten numeric digit, classify the digit from 0-9. This report provides an overview of the current implementation of this project.

Data Preprocessing

The data, is a set of handwritten digits, the size is 28x28 pixels. This data is split into four files, two files for test and training samples, and two files for test and training labels. The data is relatively evenly split between the digits from 0-9. The data will initially be in the form of a binary file, containing the grayscale value of each pixel in the image, this value will be rounded to either 1 or 0, for feature simplification. In addition, a function will be used to remove whitespace from around the digits boundaries, and enlarge the image to fill the whole space, in its new bounding box, this may distort the image in the x-direction or y-direction, but has been shown to lead to less error than leaving whitespace around the border, for SVM. Further, subsampling and max-pooling will be used to further reduce this to a 14x14 image, which will then be converted to a 196 dimension vector.

Algorithm Implementation

We are using SVM classifier by scikit-learn with automatic gamma parameter. We will be using multiple epochs to have a more fitting classifier but not a lot of epochs as to cause overfitting. We will use, 10 SVM classifiers (one for each digits), and during prediction use the one that returns the greatest cross product. We will also have to find the right C parameter so that the hyperplane has the adequate margin so that it is generalized enough but not a very narrow margin that it starts to be overfitting the dataset. We don't want C to be so tiny that even our training data is misclassified. Having a much larger C would mean that a lot of data is misclassified as well. That is why finding the right value of C parameter is important.

It is our intention to use L1 regularisation to ensure that the model is simplified to avoid overfitting and give a more general solution to the problem, though this may impact expressivity, the nature of the classification determines that this will be an effective feature to implement.

Cross Validation

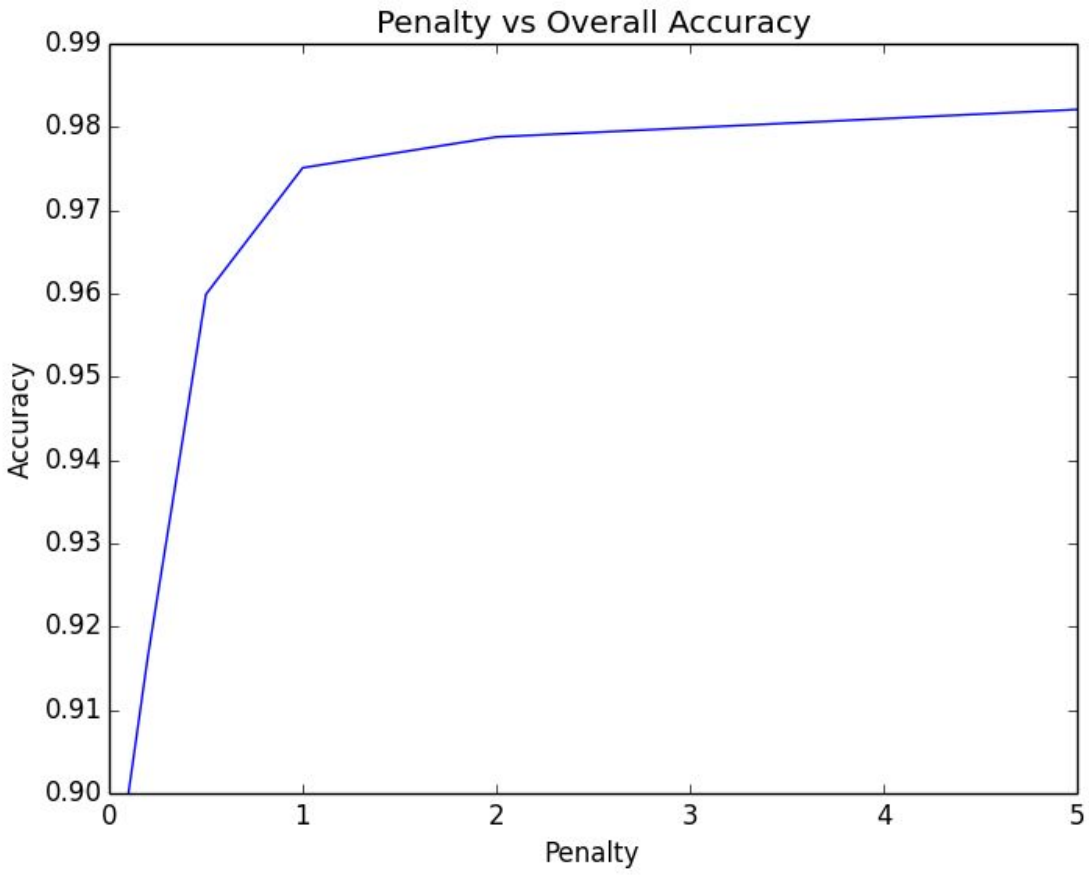
For cross-validation, K fold cross validation will be used, splitting the data into 5 sets, 4 for training and 1 for validation. Validation error, in combination with the number of epochs will be used in some cases, end training early, to prevent overfitting. We are using K fold cross validation because it will create a less biased model by making a fair split in training and testing data. The advantage is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Hyperparameter Tuning

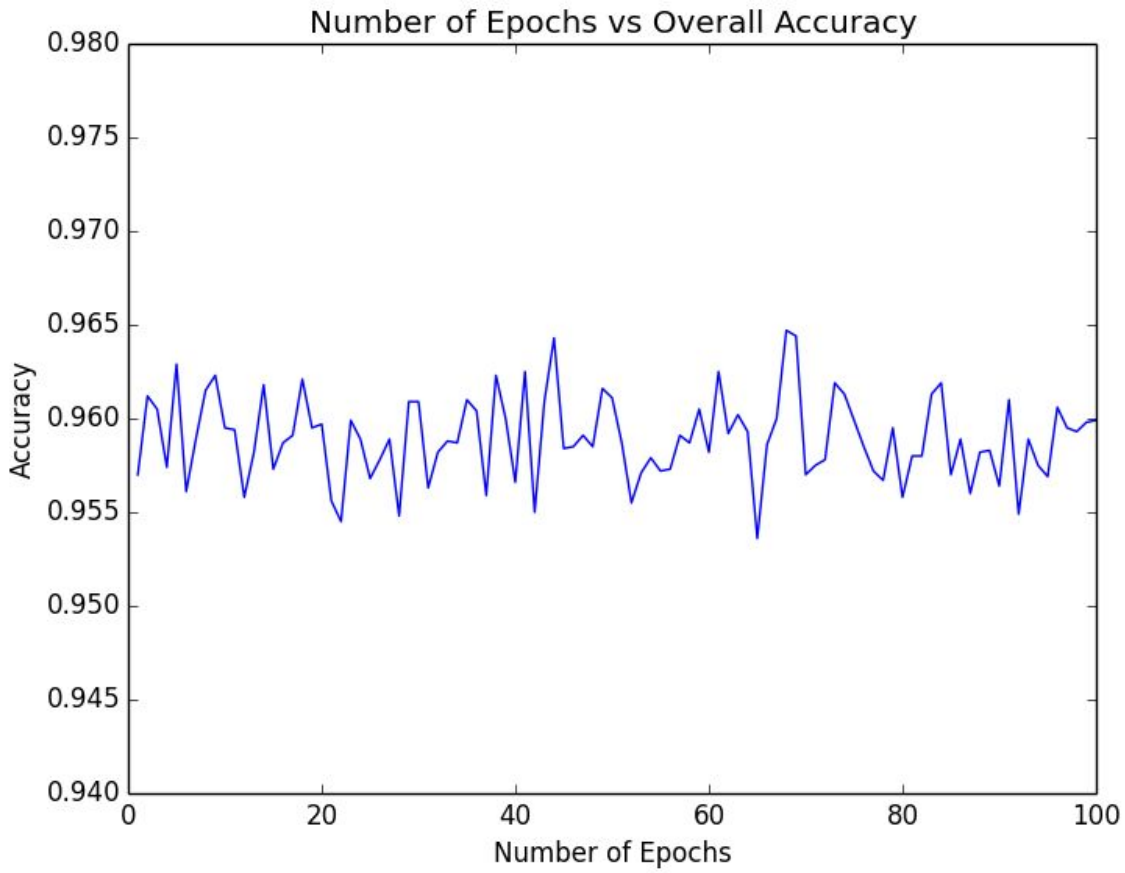
The hyperparameters to be tuned are the C penalty, learning rate, and epochs. We need to train C parameter because for large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. We need to tune the learning rates because the faster we can have the algorithm learn the classification for the 10 digits, the optimized it can be. We also need to be able to look at the learning rates for the digits separately. Doing so will make us understand which numbers are going to take longer to train and that will be a step towards improving the algorithm to accommodate for those numbers.
We are also tuning the number of epochs. With a less number of epochs, we are worried that the hyperparameter will be very biased and overfitted so we will have a trouble with the classification. On the other hand, having too many epochs will take a lot of computation and training power. For this reason, we need to find the right amount of epochs for the training dataset.

Analysis

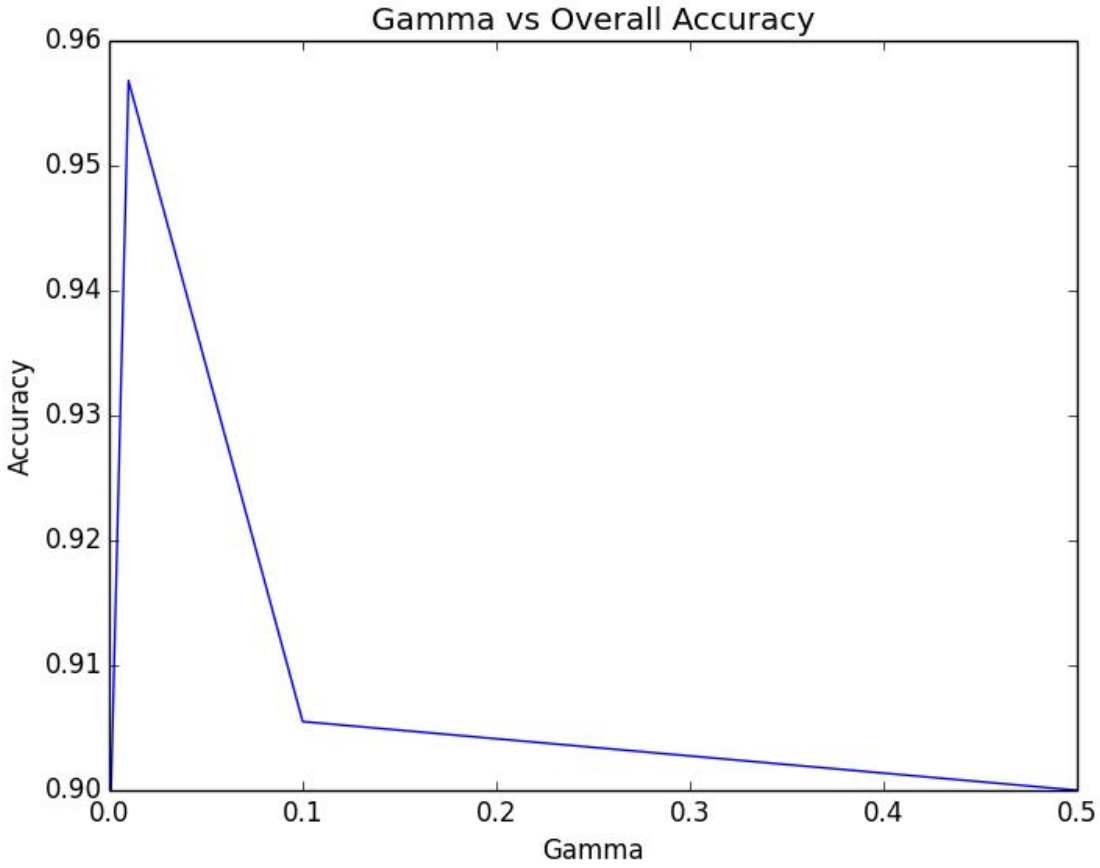
Penalty: From the graph, we found that we can ideally have a penalty of 1 or 5. A higher penalty would mean that the classifier does less generalization. A low penalty would mean the classifier will make more mistakes. A higher penalty however, is more likely to be affected by outliers, so there is a limit on how high the ideal penalty level is. In the following graph, the penalties were from [0.1, 0.2, 0.5, 1, 2, 5]. The epochs were kept constant at 1. The learning rate was also constant at 0.01.



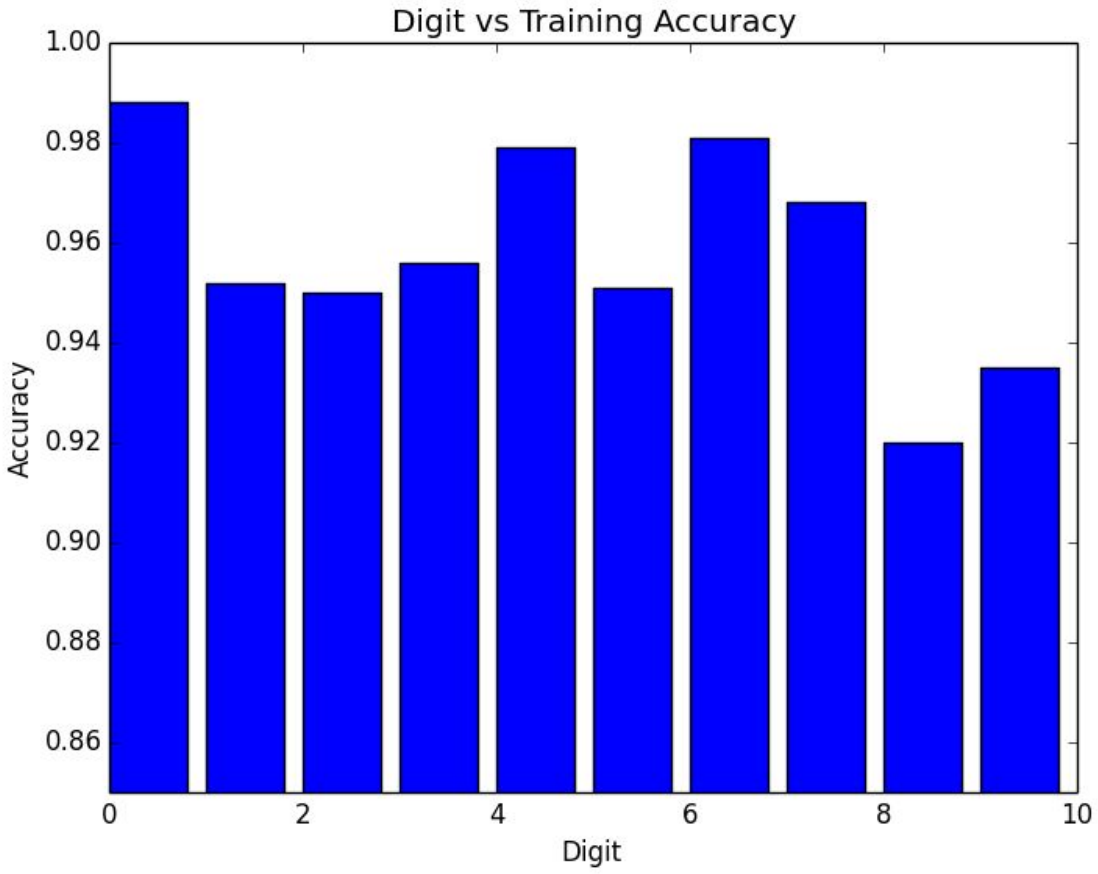
Number of Epochs: We found that the number of Epochs does not make that much of a difference in accuracy but it does take a lot of processing time for the data. This means, we should have 5-10 epochs for a more robust classifier but having a lot more will have less benefit to classification and add a lot of computation time. In the following graph, the epochs were from 1 to 100. The penalty was kept constant at 0.5. The learning rate was also constant at 0.01.



Gamma: After several tests, we found out that the ideal gamma is 0.01. Any other gamma value does not increase the accuracy but for some reason 0.01 makes the accuracy spike up. In the following graph, the learning rates were from [0.0001, 0.001, 0.01, 0.1, 0.5]. The epochs were kept constant at 1. The penalty was also constant at 0.5.



Digits and training accuracy: From the graph generated, we can see that training the classifier for 0 had the highest accuracy and training it for 8 had the lowest. After several testing, the results were consistent with the findings that 0 is the easiest to train and 8 is the hardest. In this, we had 1 epoch, 0.01 gamma and 0.5 penalty.



Conclusion

The dataset was not the easiest to work with due to high similarity between certain digits, however, the classifiers performed much better than a naive classifier, thus the error rate was acceptable and the project was a success. The use of data preprocessing had a marked effect on reducing run time, and increasing classifier expressivity. The observations and graphs were consistent with repeated iterations of testing.