Assignment 2, SENG 474

Quinn Webster

February 24, 2025

Experiments and Analysis

The aim of this project is to evaluate various machine learning models on their ability to classify images of sandals and sneakers into their respective class, after having added noise to the labels of the training dataset. To do so, we used the fashion-mnist dataset, which contains 12000 training images of sandals and sneakers, and 2000 testing images. The dataset can be found here https://github.com/zalandoresearch/fashion-mnist. Throughout this project, we experimented with various Support Vector Machine (SVM) models and neural networks, which resulted in various models with varying levels of accuracy. All models were built independently with the usage of various scikit-learn libraries.

To begin this project, we started by preprocessing the data. The Fashion-MNIST dataset contains images from 10 classes of clothing; however, we were only interested in class 5 (sandals) and class 7 (sneakers). Therefor, we first filtered the dataset to contain only these two classes and removed all other clothing images. We then normalized the images with Z-score normalization. Next, to add noise to the dataset, we randomly flipped the labels of the training set with a 0.2 probability. This will make the machine learning algorithms more likely to overfit, resulting in higher validation errors, but hopefully, lower test error scores. We are now ready to begin building models and perform analysis.

Support Vector Machines with Linear Kernel

To begin, I trained the model on the noisy training set using various values of C and tested the model on the noisy validation set. The goal was to obtain a good range of candidate C values for future experiments. Fig 1 shows the error on the noisy validation set across different C values. There is a general increase in error as C increases, with a minimum error of 26% at C = 0.001, and a maximum error at C = 8.192.

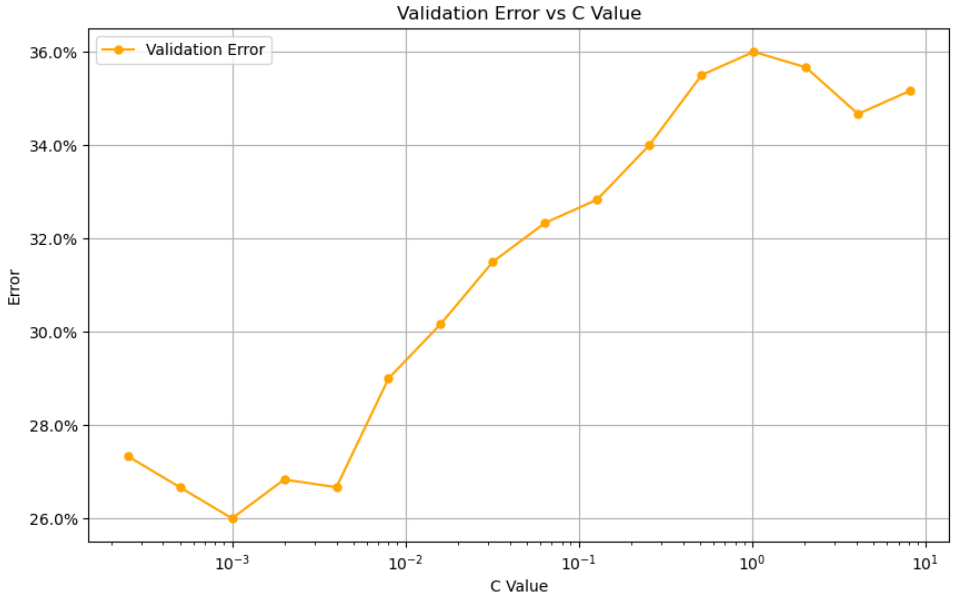

Figure 1

One reason that such a low C value results in the lowest error is due to the added label noise. Instead of trying to perfectly capture the training data (which has errors), the model instead focuses on generalization. Given the noisy labels, it is expected that the model does not achieve very low error on the validation set. These results suggest that smaller values of C result in a good balance between fitting the data well, without overfitting to the noise. This experiment results in a valuable range of C values to use in future experiments. Based on our findings, we decided to limit the values of C to a range from 0.00025 to 8.192, spaced logarithmically, for future experiments. This range of values captures both underfitting and overfitting.

Next we performed k-fold cross-validation to tune the value of C and find the optimal value that minimizes validation error. This resulted in an optimal C value of 0.0005, which gave a validation error of 25%. The results are shown in Fig 2. This result suggest that C=0.0005 reduces overfitting, while still effectively separating the data. Although 0.0005 is a small value for this parameter, due to the noise that we added to the training set, this small value helps limit the model to overfitting, and will result in lower test error in future experiments. Higher values of C seem to have caused the model to overfit due to the noisy labels. Therefor, C = 0.0005 is selected as the most effective value for future experiments.

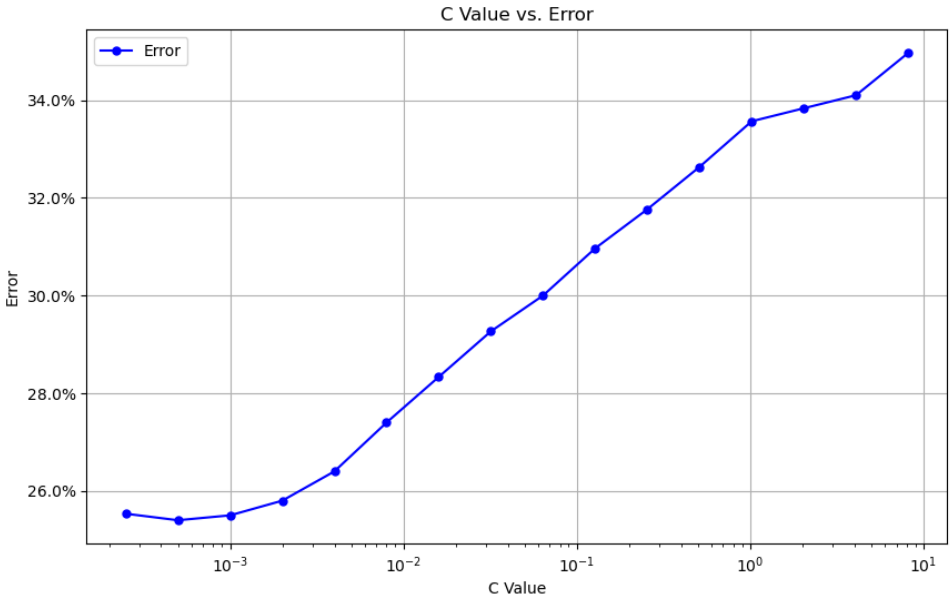

Figure 2

We then varied the values of C and trained separate models on the training data for each C value. For each model, we recorded the training error and testing error. The results (Fig 3) show that, in general, increasing the values of C leads to higher training error and testing error. With the exception of very small C values, which initially have higher error in both training and testing. These results align with our previous experiments, increasing C creates a more complex model that tries to perfectly classify each data point, including those with incorrect labs due to the added noise. Lower C values results in a simpler model that has better generalization, leading to lower test errors.

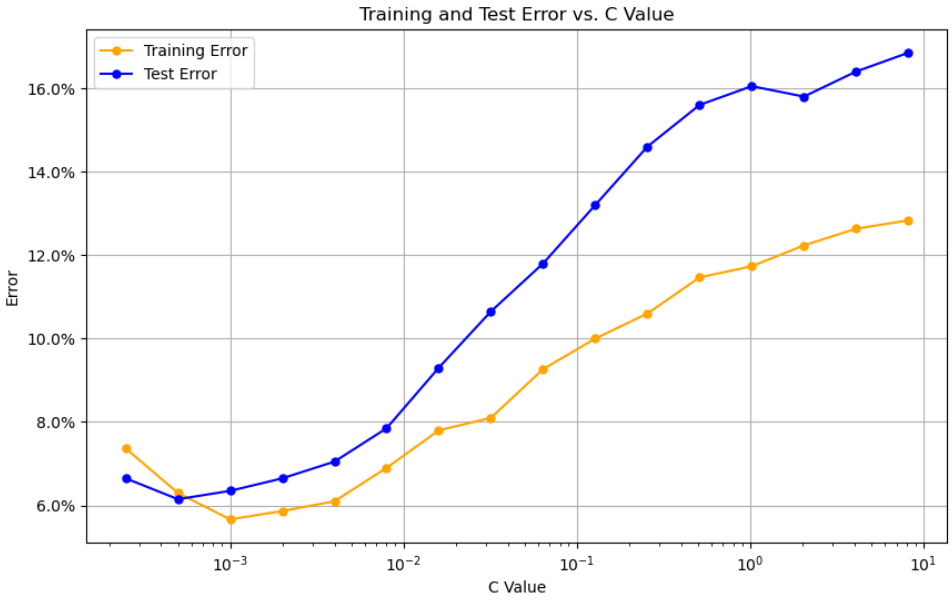

Figure 3

Support Vector Machines with Gaussian Kernel

We then started working with the Gaussian kernel for nonlinear classifiers, where we ran multiple experiments varying both y (bandwidth) and C.

First, we performed k-fold cross-validation to tune the hyperparameters C and y. For each individual y value, we found the optimal C value and then selected the combination of y and C that resulted in the lowest average error. Our result was that y = 0.001 and C = 1.28, had the lowest average error. Since y influences the shape and complexity of the decision boundary, a higher value creates a much more complex boundary, which can lead to overfitting. A smaller value like y = 0.001 finds a balance between underfitting and overfitting, allowing the model to generalize better and perform well on unseen data. The C parameter influences the trade off between training error and a smooth decision boundary. Therefore, a smaller C value results in a simpler decision boundary, but may allow some misclassified points, whereas a larger C value will do the opposite, leading to overfitting. So, with y = 0.001, and C = 1.28, we obtained a somewhat simpler model, that allows for better generalization without underfitting. This is crucial for these experiments, since we added noise to the training data, we are looking to obtain a model that has superior generalization, without getting to caught up on misclassifying the noisy training data.

Next, for each y, C pair obtained in the previous step, we trained a new model on the training dataset, and compared their training and test errors. Our results (Fig 4) show that for lower values of y (10^{-5}–10^{-2}), the test error remained quite low, ranging from 5% to 8%; however, with a y value of 10^{-1}, the testing error heavily increased to above 40%. This occurs since a higher gamma value makes the model too sensitive to individual training points (which contain noise), this leads to a very complex decision boundary that overfits the training data, and performs poorly on unseen testing data. This can be seen in the graph, as the training data remains relatively low, even while y increases, but the testing error rises rapidly.

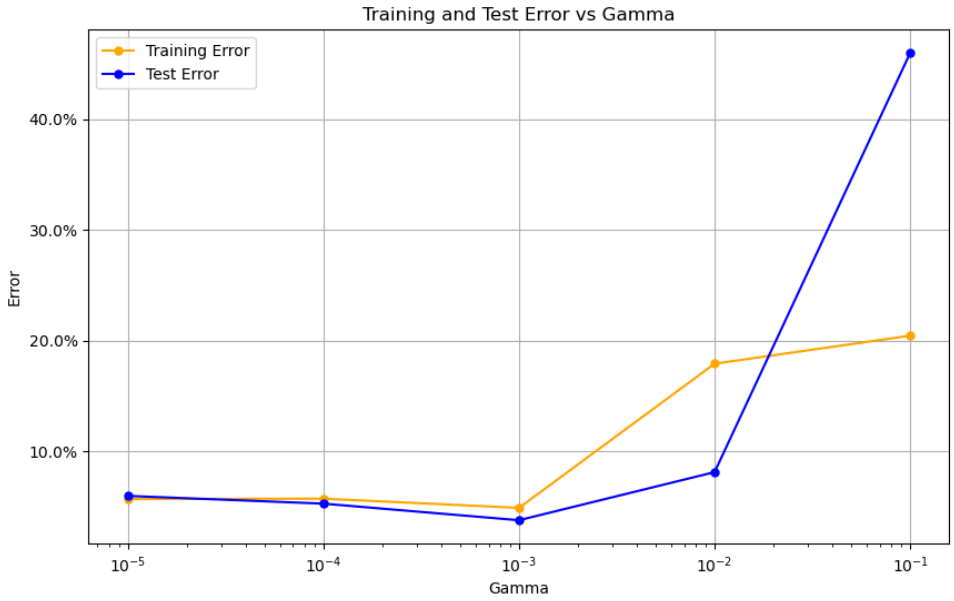

Figure 4

Neural Networks

We now begin our experiments with neural networks. To start, we performed k-fold cross validation to select the optimal hyperparameters. The hyperparameters that we explored were the number of hidden layers, the number of nodes in each hidden layer, the activation function, and the initial learning rate.

We varied the number of nodes from 30 to 50, with one and two hidden layers. We included relu and sigmoid activation functions and learning rates of 0.01 and 0.001. After performing k-fold cross validation with all possible combinations of these criteria, we obtained that the optimal configuration for a neural network was a single hidden layer with 30 nodes, using the sigmoid activation function with an initial learning rate of 0.001, this combination resulted in a validation error of 30%.

We then began experiments with varying various hyperparameters and plotting the result of the neural networks training and test error.

First, we varied the number of nodes in a single hidden layer, these results are in Figure 5. Increasing nodes from 5 to 20 led to roughly a 4% increase in training error. From 20 nodes to 80 nodes, we see no change in in the training error, which indicates that increasing the number of nodes no longer impacted the networks' ability. However, the opposite is true for the testing data. As we increase the number of nodes in the hidden layer, we see a general decrease in the testing error, this suggests that a larger neural network better captured the data without overfitting, improving the model's generalization and ability to classify unseen data.

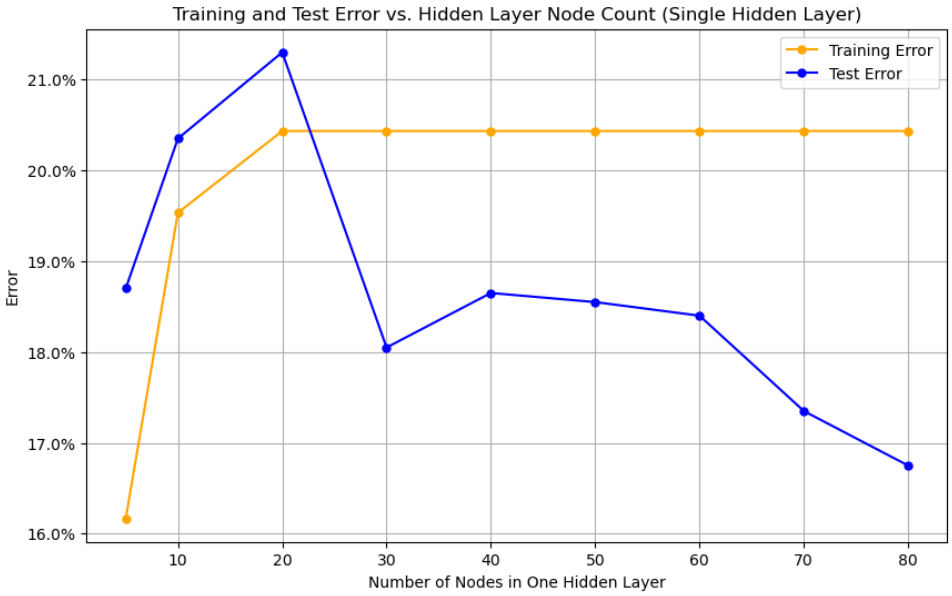

Figure 5

Next, we performed a similar experiment as above but with two hidden layers (Fig 6), where we vary the number of nodes in each (both hidden layers always have the same number of nodes). From 20 to 80 nodes in each hidden layer, there is no difference in the training error, and only a slight difference (2%) when increasing the number of nodes from 5 to 20. The test error generally decreases the more hidden nodes we have, which aligns with our results from the previous experiment.

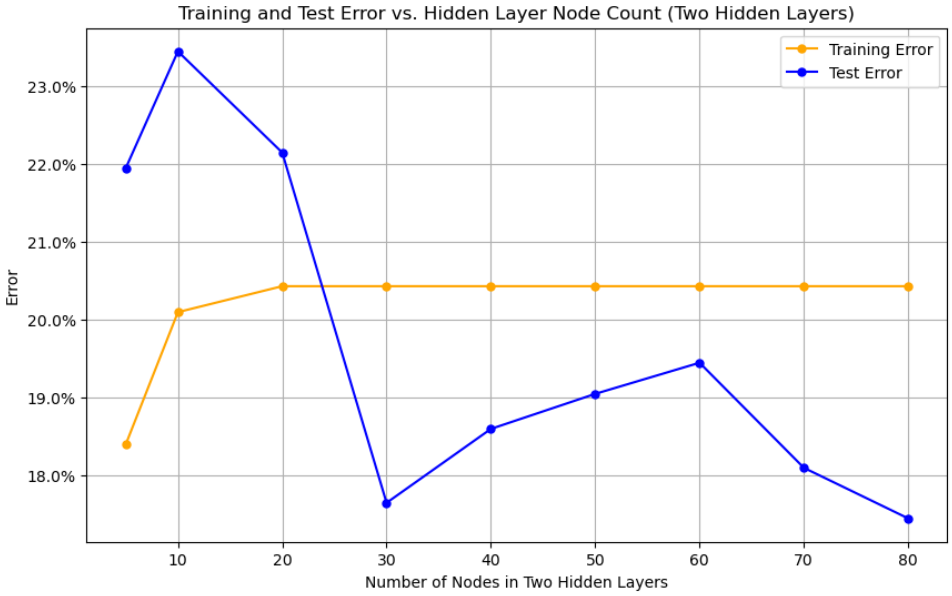

Figure 6

We then performed an experiment where we varied the number of epochs that we trained the model on (Fig 7). This dictates how many times the model is trained on the entire dataset. Both the training error and testing error followed similar trends. When the number of epochs was low (10-20), both errors are relatively low, between 8% and 10%; however, as this value is increased, so did the training and testing errors. This indicates that when there are few number of epochs, the noisy labels have less influence on the model, the model is underfitting. However, when the number of epochs is too high, the noisy labels have a heavy influence on the trained model, reducing its ability to make accurate classifications as it is overfitting.

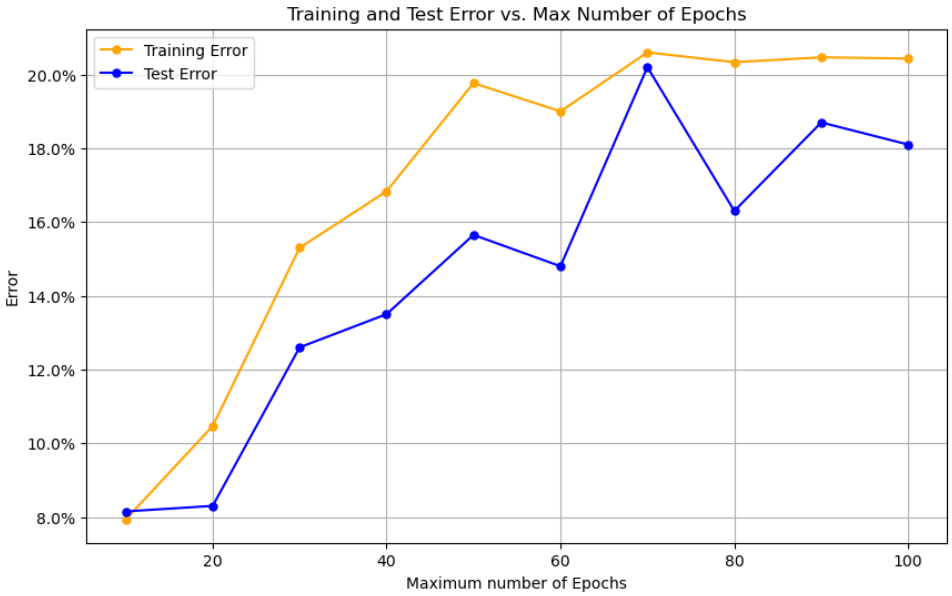

Figure 7

Finally, we tested different activation functions (relu, tanh, and sigmoid) (Fig 8). All three of these functions resulted in near identical training error values (roughly 20.5%), this indicates that activation functions have little impact on the training error for this dataset. However, for test error, relu performed the worst (18.7%), while both tanh and sigmoid obtained lower errors (around 16%). This suggests that tanh and sigmoid may offer better generalization for this specific problem, this could be because of the variations in the training data and is not necessarily mean that tanh performs far superior to the other functions. In future experiments, I would run models with various activation functions to ensure that I am using one that results in the best values.

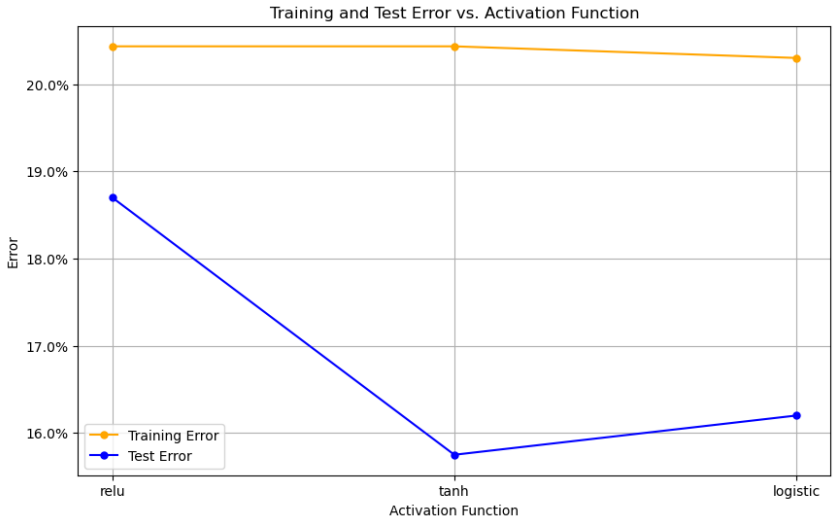

Figure 8

Model Comparison: Linear SVM, Gaussian Kernel SVM, and Neural Network

From the previous experiments, we obtained the best hyperparameters for each model:

- **Linear SVM**: $C = 0.0005$
- **SVM with Gaussian Kernel**: $\gamma = 0.001$, $C = 1.28$
- **Neural Network**: One hidden layer with 30 nodes, sigmoid activation function, and an initial learning rate of 0.001.

The SVM with the Gaussian kernel outperformed both the linear SVM and the neural network, obtaining the lowest test error of 3.55%. This suggests that the dataset that we used had non-linear relationships between datapoints that the non-linear Gaussian kernel effectively captured. The linear SVM, performed also performed quite well, with a test error of 5.5%. This results, combined with the results from the Gaussian kernel, indicates that although the data likely has non-linear features, using a linear model can still effectively capture the patterns in the dataset and form accurate classification after. The neural network had the highest test error at 14.2%, although we performed k-fold cross-validation to obtain the most optimal set of parameters, perchance, we did not tune a wide enough range of hyperparameters. In future experiments, a broader search of parameters may lead to a more effective configuration. The noisy labels may have also impacted the neural networks classification ability, while training, it may have learned the noise to effectively, therefor, forming less accurate classifications on unseen data.

Data Augmentation

In this section, we experimented with using a very small training set of only 20 datapoints, 10 from each class. We also used a small validation set of 100 datapoints, 50 from each class. We then performed various experiments with augmenting the dataset with random image rotations. How this worked was we has a parameter m, which set how many new images we created for each original image, and a parameter r, which defined the range of degrees from [-r, r] that we could randomly rotate an image. The goal of these experiments was to test the ability of a Support Vector Machine with the Gaussian Kernel to classify images after only being trained on the augmented small training set.

First, as a baseline model, we tuned the parameters C (the regularization parameter) and y (the bandwidth parameter) on the original small training set without augmentation. We then computed the test error on the tuned model. This resulted in a test error of 37%, which make sense with the small size of the training set. In fact, this is actually somewhat impressive, as with only 20 training datapoints, the model was able to correctly classify 1260 of 2000 (63%) test examples. This gave us a baseline to see how much better our augmented training set could perform.

We then ran an experiment where we tuned the following parameters

- **m: the number of augmented examples per original example**
- **r: the range parameter**
- **y: the bandwidth parameter**
- **C: the regularization parameter**

We performed an extensive grid search where we used the validation set to compare model errors. The results from this experiment were that the optimal values of m, r, y, and C were 20, 5, 0.0001, and 1 respectively. Using these values, we then trained a final model on the augmented training data with these parameters and received a testing error of 12.3%. This is quite an improvement from our baseline test, an improvement of 24.7%, while only augmented versions of the same small dataset, no additional datapoints were added. This demonstrates the value that data augmentation stands to offer in machine learning. By augmenting the dataset with random rotations of the original data, we developed a model that is more generalized and results in better classification.

We then ran an experiment where we used a fixed value of r = 10, which allows for some rotation of images, but not any such rotation that seriously alters the images appearance. We then tuned the values of y and C for each value of m. After having gathered the optimal values of y and C for each m, we then trained models on the augmented training set for each m with its respective y, C pair, and plotted the results (Fig 9). With a range of values from 10 to 100, there is only a difference of 1.2% in testing error, suggesting that the value of m does not play a significant role in reducing the test error, as long as we are still adding some augmented data to the training set. There seems to be a slight downward trend , with a large spike m = 70, this may be due to slight overfitting of the model, yet nothing drastic enough to heavily influence its classification ability (the difference is still around 1%). Overall, my takeaway from this experiment is that as long as we are adding augmented data to the training set, the models classification ability significantly improves.

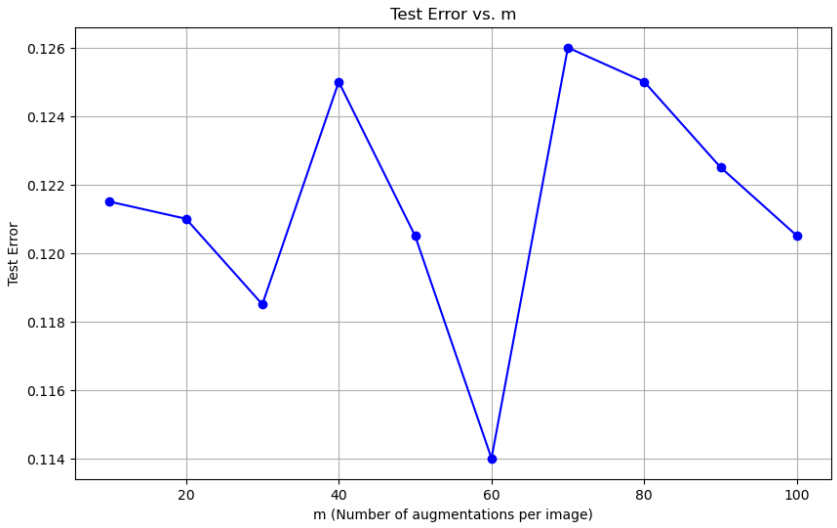

Figure 9

We performed a similar experiment as above where we kept m constant at 30, which allows for a fair number of augmented images, without making the training set very large. While varying the value of r, we found the optimal values of y and C for each r. We then trained a model on each value of r with its tuned y, C pair. The results are shown in Fig 10. These results align with our previous experiments, that as r increases, so does the test error. This likely happens because a larger value of r results in greater rotation, which causes greater variance in between images, which may lead to misclassification. Although, it is important to note that as r ranges from 10 to 100, the test error only varies approximately 4.5%. This suggests that although greater values of r may lead to higher test errors, the model remain quite accurate.

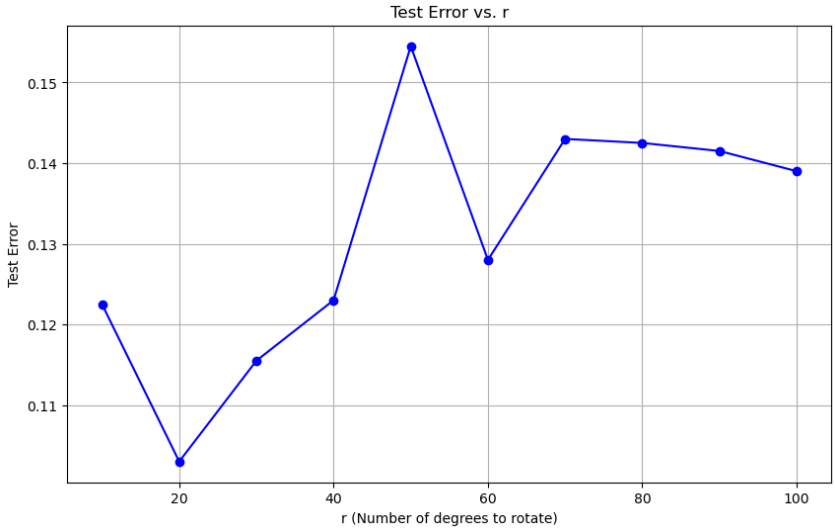

Figure 10