1. Problem 1. Regression

1.1 Data Pre-Processing

The dataset involves socio-economic data from the 1990 US census for various US communities and their corresponding number of violent crimes per capita. Standardisation was applied to the data before the implementation of lasso and ridge regression, as these regression methods penalise coefficient magnitudes associated with each variable and the scale of these variables affect the amount of penalisation applied on their respective coefficients (Jaadi, 2019). High variance variable coefficients are small and are therefore penalised to a lesser degree and vice versa. Standardisation ensures that all features are on the same scale, which in turn increases the performance of both lasso and ridge regression models as it aids their ability to choose important features and decreases the magnitude of model coefficients (Pluviophile, 2022). It also helps us to visualise the data easier as it transforms said data to a standard normal distribution.

1.2 Model Details

Three models were trained on the socio-economic data, namely a linear regression model, ridge regression model, and lasso regression model. All models were evaluated and fitted on the testing dataset, and constant terms were added for the linear regression model. This model displays the coefficients and their corresponding statistics (such as P-values) which affect our dependant variable of violent crimes per capita, as well as values to determine model accuracy (such as R^2). For the lasso model, 1000 values of λ between zero and 0.25 were tested on the validation dataset. For the ridge model, 50 values of λ between zero and 1.0 were tested. An evaluation function will be implemented to automatically test each of these λ values on the data and evaluates which of those λ values present the best values of R^2, adjusted R^2, RMSE, and the coefficients of the standardised dataset, as well as display plots which visually aid in selecting an optimised value of λ for the dataset. The aforementioned limits of values of λ to be tested for the lasso and ridge models were chosen based on initial testing done to see where the general value λ would be based on the resulting RMSE vs λ graph. The limits are different between lasso and ridge as an optimised value for one model does not necessarily mean it will go as well on the other model, as one deals with the sum of squares and the other with sum of absolute values. Limiting to between 0-0.25 and 0-1 for the ridge and lasso models respectively will give a better visualisation of the optimal value of λ for the respective models.

1.3 Model Evaluations

Figure 1 in the appendix shows the results of the linear regression model fitted onto the socio-economic data. An R^2 value of 0.784 indicates that our model explains 78.4% of the variability observed in the 'violent crimes per capita' dependant variable, which is evidence to believe our model is a reasonably good fit. The RMSE value for the testing data is 0.106. The coefficients presented in Figure 2 though present a few problems. A large amount of our coefficients seems to look statistically insignificant, or those with a P-value greater than 0.05. This suggests that removing these coefficients from the model may improve its performance, though due to the substantial number of coefficients to remove, as well as considering the socio-economic nature of the data, this was not done. The Q-Q plot in Figure 3 shows a somewhat linear fit, indicating the variance within the data samples are relatively common, and therefore the dataset is homoscedastic. Running our model on the test data presents Figure 4, which shows a reasonably adequate ability to predict violent crimes per capita, though at times underestimates the magnitude of troughs as well as misrepresenting some of the peaks in the actual data.

Figure 5 displays the result of the ridge regression model ran on the testing set. We see that we get a best R^2 value of 0.72 based on the values of lambda chosen, which is lower than given in the linear regression model. Our best RMSE on the testing set is 0.156 which is slightly worse than of the linear regression model. The RMSE vs lambda plot immediately shows how regularisation helps the model as the performance increases instantly (RMSE in the y-axis decreases) as lambda increases. Though, the tail end of the validation RMSE of the plot seems to start increasing, indicating the model is starting to over-regularise as lambda increases above 0.25. Our best lambda appears to be in the range of 0.05 and 0.15 for a ridge model. Visually, running the model on the testing set, ridge appears to be more accurate in comparison to the linear model, with it no longer underestimating the magnitude of troughs and reasonably predicting the peaks better. The ridge Q-Q plot also appears to follow the Y=X line more linearly than the linear model, and the final trace plot behaves as expected, whereas we increase lambda, our coefficients gradually tend towards 0 though never reaches 0. Overall, our ridge model statistically fits slightly worse than the linear model, though it seems to do a better job at predicting the violent crimes per capita based on the visual plots.

Figure 6 displays the result of the lasso regression model fitted onto the socio-economic data. We get a best R^2 value and best RMSE on the testing set of 0.73 and 0.14 respectively, which very slightly improve on the ridge model though are still statistically worse than the

linear model. Our plots however visually tell the same story as ridge does, with it being visually an improvement over the linear model. Interestingly, the RMSE vs lambda plot shows that RMSE does not get better as lambda increases using the lasso model as was seen on the ridge model. The test set performance and Q-Q plots has similar trends to the ridge model, improving on the linear model. The trace plot shows that a lot of the coefficients go to zero at some point around a lambda of 0.7. Overall, the lasso model very slightly improves on the ridge model though still does not fit as well as the linear model on the socio-economic dataset.

Something to note is the socio-economic aspect of the data. Even with the extent of variables within the dataset, it does not consider, among other further aspects, the political leaning of these communities, which may explain at least some of the discrepancies between actual and predicted values and potential outliers. The following table summarises some of the statistical results of the linear, ridge, and lasso models.

	R-squared	Adjusted R-squared	RMSE (testing)
Linear	0.784	0.675	0.106
Ridge	0.72	0.577	0.156
Lasso	0.73	0.596	0.14

Table 1: Linear, Ridge, Lasso model results

1.4 Ethical Concerns on Socio-Economic Modelling

It is important to consider biases that may be present within the data, potentially affected by how the data was gathered for example. Bias may also present itself in terms of analysing the data, and this is especially harmful with socio-economic data, as potentially inaccurate predictions may disproportionately affect more socially vulnerable of marginalised groups, such as through discrimination. With the data being sourced from the 1990s, the algorithms trained may create decisions based on past, historic biases. Therefore, we must take care in making decisions based on these algorithms to avoid misinterpretations and repeating these historical biases, especially when potentially applying these decisions to modern life more than three decades into the future.

Problem 2. Classification

2.1 Data Pre-Processing

Figure 7 shows the boxplot of the raw data including 27 spectral properties and overall classification of land type, which may be: *s* – 'Sugi' forest, *h* – 'Hinoki' forest, *d* – 'Mixed deciduous' forest, and *o*: 'Other' non-forest land. An imbalance can be seen between dimensions (spectral properties), with some dimensions having a much larger range than others. Particularly, dimensions 8, 17, 20, 21, 23, 24, 26, and 27 have tiny values. As SVMs and CKNNs rely on distances, it is imperative that we standardise the dataset as the imbalance may distort their results. Note the range imbalance issue is mostly solved with the standardised data boxplot shown in Figure 8. Though, we do still see some class imbalance as shown in Figure 9, where some classes (forests) are underrepresented particularly in the testing set. A positive is that we have all classes in both the training and testing sets, though they do not particularly share a similar overall distribution to each other.

2.2 Model Hyper-Parameters

The dataset was trained on three multi-class classifiers that classify land type from the spectral data. Each classifier model's hyper-parameters were selected using a grid search operating over the validation set. The first model is Continuous K Nearest Neighbours (CKNN), whereby testing on the 'n_neighbors' hyper-parameter value was done. This hyper-parameter denotes the number of neighbours to use for the model. A smaller number means a more precise separation of classes at the expense of a greater sensitivity to noise, and a larger number means a smoother boundary between the land type classes, though may cause problems when some classes are scarce. Using a grid search function, 'n_neighbors' values from 1 to 64 were tested.

The second model is a Random Forest, whereby testing on the 'n_estimators', 'max_depth', and 'class_weight' hyper-parameters were done. As the 'n_estimators' increase, the variation of the model will decrease. As 'max_depth' increases, the class boundary complexity and risk of overfitting increases. A good value of depth must consider all classes. Three values of 'class_weight' were tested: 'balanced' which uses values of y to automatically adjust weights inversely proportional to class frequencies in the data, 'balanced_subsample' which is similar to 'balanced' except weights are computed based on bootstrap sampling for each tree grown, and no weighting used (Pedregosa, et al. 2011). A grid search function will be implemented to test each of these hyper-parameters.

Lastly, we have an ensemble of Support Vector Machines (SVMs), whereby testing on the 'C', 'kernel', 'degree', and 'gamma' hyper-parameters were done. The strength of regularisation is inversely proportional to the 'C' regularisation hyper-parameter. 'kernel' specifies which kernel type is used in the algorithm and is a method of using a linear classifier to solve a non-linear problem. Kernels to be used are linear kernels, radial basis function (rbf) kernels, and polynomial kernels. 'degree' specifies the degree if a polynomial kernel is specified. 'gamma' is the kernel coefficient of an 'rbf', 'poly', and/or 'sigmoid' kernel if chosen. A grid search function will be implemented to test each of these hyper-parameters.

2.3 Model Evaluations

Figure 10 displays the results of doing a grid search over the validation set to find the optimal value of 'n_neighbors' for a CKNN model. From this, we see that the optimal value is 3. We may then evaluate the model with the 'n_neighbors' hyper-parameter set to 3 over the testing set, which we see the results of in Figure 11. These results are reasonably positive, as we obtain high precision, recall, and F1 scores for all our classes, meaning our model has a good positive predictive value and a high sensitivity or true positive rate (lesser amounts of false positives and false negatives). The F1 score is the harmonic mean of precision and recall. With our model accuracy of 0.91, and viewing the testing set performance plot in the same figure, we can deduce that a CKNN with an 'n_neighbors' hyper-parameter set to 3 produces a model that is highly accurate that fits well over all the land type classifications.

The grid search to find the optimal hyper-parameters for a random forest model can be seen in Figure 12. From this, we can see that the best hyper-parameters for a random forest is a 'class_weight' of 'balanced', a 'max_depth' of 3, and 25 'n_estimators'. The result of this model evaluated on the testing set is displayed in Figure 13. The results are quite positive as well, with high precision, recall, and F1 scores across the board. With a model accuracy of 0.88 and viewing the testing set performance plot, this random forest model with these hyper-parameters performs very well, though slightly less than the CKNN model. Though, with further fine-tuning of hyper-parameters, especially evaluating over a higher value of 'max_depth' without overfitting further, may edge the model over CKNN.

Lastly, we have a grid search to find optimal hyper-parameters for an ensemble of SVMs in Figure 14. Figure 15 evaluates that grid search onto the testing set. From these, we deduce that the bets hyper-parameters for an ensemble of SVMs is a 'C' of 0.1, 'degree' of 6, 'gamma' of 0.1, and a polynomial kernel. The results of this model generally do not go as well

as either CKNN or random forest. We see F1 scores ranging from mediocre to reasonably high (0.57 – 0.72), and an overall model accuracy of 0.70 which is not terrible. Depending on the land type coefficient we see either perfect or middling precision and recall scores. Overall, the ensemble of SVMs performs the worst out of all the models.

Problem 3. Training and Adapting Deep Networks

3.1 Neural Network Design and Training Approach

The MNIST-style dataset provides an image collection of Street View House Numbers (SVHN) roughly centred in the middle of the image. These images are loaded at size 32 x 32 x 3 channels (colour image) and initially without augmentation as there is a reasonable amount of data (1000 training samples and 10000 test samples). Though, augmentation will be exercised later to observe its effects on the data. Figure 16 shows some of the non-augmented images of the house numbers. A VGG-like neural network was trained and used to evaluate the dataset. The network used to train the non-augmented images is as follows: 1. an input layer with a 32 x 32 x 3 shape, 2. a 3 x 3 convolution block consisting of two convolutional layers with 8 filters separated by a ReLU activation layer then applying a 2 x 2 max pooling layer, 3. the same 3 x 3 convolution block but with 16 filters per convolutional layer, 4. the same 3 x 3 convolution block but with 32 filters per convolutional layer, 5. flattening layer, 6. a dense layer with 512 neurons with a ReLU activation, 7. an output layer with 10 neurons for the 10 possible number classes with a softmax activation. The models for non-augmented and augmented data are shown in figures 18 and 19. Due to limited computational resources, the 8, 16, and 32 filters were used, as higher filter values presented crashing issues to the hardware. The network used to train the augmented images is similar though we change the input layer to take in a 50 x 50 x 1 shape. The model is compiled using a categorical cross entropy loss function and Adam optimiser. The non-augmented network has 285,970 total parameters, while the augmented network has more than twice the amount at 613,506 total parameters.

3.2 Data Augmentations

The augmentations used includes resizing the image to 50 x 50, converting the images to grayscale (1 channel), small random rotations up to 5% each side, as well as vertical and horizontal shifts up to 5%. The images were resized to improve the fidelity of the image and potentially make the numbers clearer in the image. The grayscale conversion was done to further make the numbers clearer against the house wall backgrounds. Some of the numbers were not fully centred and even included extra numbers beside it. The vertical and horizontal shifts were done to capture the number as intended as well as potentially capturing the neighbouring numbers to increase the amount of data to train over. Figure 17 shows the augmented images of the house numbers.

3.3 Neural Network and SVM Evaluations

The SVM fitted on the data is shown in Figure 22 and uses a 'C' of 1 and a linear kernel. We observe reasonably bad results in terms of precision, recall and F1 scores, and an overall accuracy of 0.31, indicating the model does not perform well on the testing set. This can be attributed to a few reasons, such as SVMs being potentially unsuitable for larger datasets (the SVHN testing set having 10000 images), or the data may have some class imbalances (Khoong, 2021). We will keep this in mind as we evaluate the network on the non-augmented and augmented data.

The network was trained with a batch size of 128 and 32 epochs for the non-augmented data and 64 epochs for the augmented data. The results of these trainings are shown in figures 20 and 21 for non-augmented and augmented, respectively. The non-augmented data was trained for 36 seconds, but we observe overfitting as the testing loss goes backward even as its accuracy increases and begins to converge at 32 epochs. It took around 15-16 epochs for the testing loss of the non-augmented data to begin going backwards. The augmented data was trained for 160 seconds, and again we observe overfitting with testing loss starting to go backwards at around 30 epochs, though the accuracy has converged at around 0.77 at around 39 epochs. It is important to note the random nature of the neural networks, and future results may differ from those presented above.

Confusions charts and F1 scores for both networks on the testing sets are shown in figures 23 and 24. The testing set F1 scores of the non-augmented and augmented data are 0.695 and 0.813, thus we can deduce that the VGG-like network for the augmented data performs significantly better than the non-augmented network. The figures present a potential class imbalance within the dataset, such as there being more lower numbers than higher numbers, though it does not seem to cause a major impact on the model's ability to predict said numbers, and there seems to be a low amount of confusion between numbers predicted and actual.

Overall, we see that the augmentations of resizing the images, converting them to grayscale, and applying horizontal and vertical shifts to the data improves the model's ability to predict the house numbers. There is a potential to further improve the model by addressing class imbalance issues as mentioned within the data.

Appendix

```
Linear Model Validation Data: RMSE = 0.15426186711426265
Linear Model Testing Data: RMSE = 0.10648954801192673
                            OLS Regression Results
==============================================================================
Dep. Variable:      ViolentCrimesPerPop    R-squared:                       0.784
Model:                              OLS    Adj. R-squared:                  0.675
Method:                   Least Squares    F-statistic:                     7.200
Date:                  Sun, 16 Apr 2023    Prob (F-statistic):           3.93e-32
Time:                          18:38:44    Log-Likelihood:                 245.41
No. Observations:                   299    AIC:                            -288.8
Df Residuals:                       198    BIC:                             84.92
Df Model:                           100
Covariance Type:                nonrobust
```

Figure 1: Linear Regression Model Results

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.1841	0.726	-0.254	0.800	-1.616	1.248
population	0.0930	1.535	0.061	0.952	-2.934	3.120
householdsize	-0.0211	0.341	-0.062	0.951	-0.693	0.651
racepctblack	0.3181	0.172	1.854	0.065	-0.020	0.656
racePctWhite	0.1887	0.211	0.857	0.393	-0.235	0.597
racePctAsian	-0.0620	0.105	-0.777	0.438	-0.270	0.126
racePctHisp	-0.1245	0.184	-0.676	0.500	-0.488	0.239
agePct12t21	-0.0932	0.342	-0.273	0.785	-0.767	0.581
agePct12t29	0.0882	0.463	0.173	0.863	-0.834	0.994
agePct16t24	-0.2241	0.509	-0.441	0.660	-1.227	0.779
agePct65up	0.0181	0.319	0.057	0.955	-0.610	0.647
numbUrban	-0.3426	1.425	-0.241	0.810	-3.152	2.467
pctUrban	0.0261	0.051	0.507	0.612	-0.075	0.128
medIncome	-0.3725	0.597	-0.624	0.533	-1.550	0.805
pctWage	0.0875	0.277	0.316	0.753	-0.459	0.634
pctWFarmSelf	0.0425	0.062	0.690	0.491	-0.079	0.164
pctWInvInc	0.0311	0.216	0.144	0.885	-0.394	0.457
pctWSocSec	0.0653	0.325	0.201	0.841	-0.577	0.707
pctWPubAsst	0.0665	0.174	0.382	0.703	-0.277	0.409
pctWRetire	-0.0415	0.127	-0.327	0.744	-0.292	0.209
medFamInc	0.5054	0.587	0.860	0.391	-0.653	1.664
perCapInc	-1.3630	0.636	-2.144	0.033	-2.617	-0.109
whitePerCap	1.1407	0.552	2.065	0.040	0.051	2.230
blackPerCap	0.0261	0.077	0.338	0.736	-0.126	0.178
indianPerCap	-0.0561	0.058	-0.962	0.337	-0.171	0.059
AsianPerCap	0.1012	0.054	1.863	0.064	-0.006	0.208
OtherPerCap	0.0719	0.046	1.561	0.120	-0.019	0.163
HispPerCap	-0.0411	0.076	-0.538	0.591	-0.192	0.110
NumUnderPov	0.5426	0.495	1.097	0.274	-0.433	1.518
PctPopUnderPov	-0.3046	0.215	-1.419	0.157	-0.728	0.119
PctLess9thGrade	-0.1746	0.202	-0.863	0.389	-0.573	0.224
PctNotHSGrad	0.0192	0.262	0.073	0.942	-0.498	0.536
PctBSorMore	-0.2595	0.240	-1.079	0.282	-0.734	0.215
PctUnemployed	-0.0578	0.145	-0.399	0.690	-0.343	0.228
PctEmploy	-0.2124	0.284	-0.748	0.456	-0.773	0.348
PctEmplManu	-0.1417	0.092	-1.532	0.127	-0.324	0.041
PctEmplProfServ	-0.0288	0.132	-0.218	0.828	-0.290	0.232
PctOccupManu	0.2602	0.148	1.757	0.080	-0.032	0.552
PctOccupMgmtProf	0.4703	0.289	1.628	0.105	-0.099	1.040
MalePctDivorce	0.5367	0.892	0.602	0.548	-1.221	2.295
MalePctNevMarr	0.3479	0.215	1.617	0.107	-0.076	0.772
FemalePctDiv	0.4033	1.192	0.389	0.698	-1.886	2.813
TotalPctDiv	-0.7871	1.968	-0.400	0.690	-4.667	3.093
PersPerFam	-0.0766	0.505	-0.151	0.880	-1.073	0.920
PctFam2Par	0.1308	0.546	0.240	0.811	-0.945	1.207
PctKids2Par	-0.2381	0.518	-0.459	0.646	-1.260	0.784
PctYoungKids2Par	-0.3133	0.163	-1.920	0.056	-0.635	0.008
PctTeen2Par	-0.0392	0.138	-0.283	0.777	-0.312	0.234
PctWorkMomYoungKids	0.0471	0.149	0.315	0.753	-0.247	0.342
PctWorkMom	-0.1408	0.163	-0.866	0.387	-0.461	0.180
NumIlleg	-0.5364	0.375	-1.430	0.154	-1.276	0.203
PctIlleg	-0.0003	0.175	-0.458	0.648	-0.426	0.266
NumImmig	-0.1301	0.322	-0.405	0.686	-0.765	0.504
PctImmigRecent	0.1121	0.102	1.095	0.275	-0.090	0.314
PctImmigRec5	-0.3083	0.189	-1.631	0.104	-0.681	0.064
PctImmigRec8	0.0402	0.268	0.150	0.881	-0.488	0.568
PctImmigRec10	0.1846	0.214	0.862	0.390	-0.238	0.607
PctRecentImmig	-0.3699	0.428	-0.864	0.389	-1.214	0.475
PctRecImmig5	-0.1674	0.846	-0.198	0.843	-1.836	1.501
PctRecImmig8	0.8263	1.114	0.742	0.459	-1.370	3.023
PctRecImmig10	-0.3275	0.962	-0.340	0.734	-2.225	1.570
PctSpeakEnglOnly	0.0314	0.226	0.139	0.889	-0.414	0.477
PctNotSpeakEnglWell	0.2390	0.276	0.867	0.387	-0.304	0.782
PctLargHouseFam	-0.2442	0.752	-0.325	0.746	-1.726	1.238
PctLargHouseOccup	-0.1926	0.801	-0.241	0.810	-1.772	1.386
PersPerOccupHous	0.3631	0.794	0.457	0.648	-1.203	1.929
PersPerOwnOccHous	0.2586	0.502	0.515	0.607	-0.731	1.248
PersPerRentOccHous	0.2021	0.251	0.806	0.421	-0.292	0.697
PctPersOwnOccup	0.0318	1.157	0.028	0.978	-2.251	2.314
PctPersDenseHous	0.5137	0.289	1.779	0.077	-0.056	1.083
PctHousLess3BR	0.2343	0.184	1.271	0.205	-0.129	0.598
MedNumBR	0.0263	0.058	0.451	0.653	-0.089	0.141
HousVacant	0.1000	0.242	0.446	0.656	-0.369	0.585
PctHousOccup	-0.1888	0.095	-1.991	0.048	-0.376	-0.002
PctHousOwnOcc	-0.0010	1.200	-0.001	0.999	-2.367	2.366
PctVacantBoarded	0.0450	0.066	0.684	0.495	-0.085	0.175
PctVacMore6Mos	-0.0716	0.092	-0.782	0.435	-0.252	0.109
MedYrHouseBuilt	-0.0050	0.098	-0.051	0.959	-0.198	0.188
PctHousNoPhone	0.0019	0.113	0.017	0.986	-0.221	0.225
PctWOFullPlumb	-0.0812	0.064	-1.270	0.205	-0.207	0.045
OwnOccLowQuart	1.0232	0.810	1.264	0.208	-0.574	2.620
OwnOccMedVal	-1.5777	1.223	-1.290	0.199	-3.990	0.835
OwnOccHiQuart	0.6458	0.587	1.100	0.273	-0.512	1.803
RentLowQ	-0.1577	0.218	-0.722	0.471	-0.588	0.273
RentMedian	0.1010	0.551	0.183	0.855	-0.985	1.187
RentHighQ	-0.1496	0.273	-0.549	0.584	-0.687	0.388
MedRent	0.1629	0.397	0.410	0.682	-0.621	0.946
MedRentPctHousInc	0.1161	0.095	1.228	0.221	-0.070	0.303
MedOwnCostPctInc	-0.0652	0.109	-0.599	0.550	-0.280	0.150
MedOwnCostPctIncNoMtg	-0.1199	0.078	-1.547	0.124	-0.273	0.033
NumInShelters	0.0302	0.186	0.162	0.871	-0.337	0.397
NumStreet	0.1473	0.122	1.210	0.228	-0.093	0.388
PctForeignBorn	-0.1129	0.322	-0.351	0.726	-0.747	0.522
PctBornSameState	0.1472	0.125	1.173	0.242	-0.100	0.395
PctSameHouse85	0.1827	0.195	0.936	0.350	-0.202	0.568
PctSameCity85	-0.0971	0.123	-0.787	0.432	-0.340	0.146
PctSameState85	-0.0853	0.139	-0.614	0.540	-0.359	0.189
LandArea	0.0197	0.221	0.089	0.929	-0.417	0.456
PopDens	0.1151	0.096	1.205	0.230	-0.073	0.303
PctUsePubTrans	0.0332	0.076	0.439	0.661	-0.116	0.182
LemasPctOfficDrugUn	0.0321	0.057	0.561	0.575	-0.081	0.145

Figure 2: Coefficients for Linear Regression Model

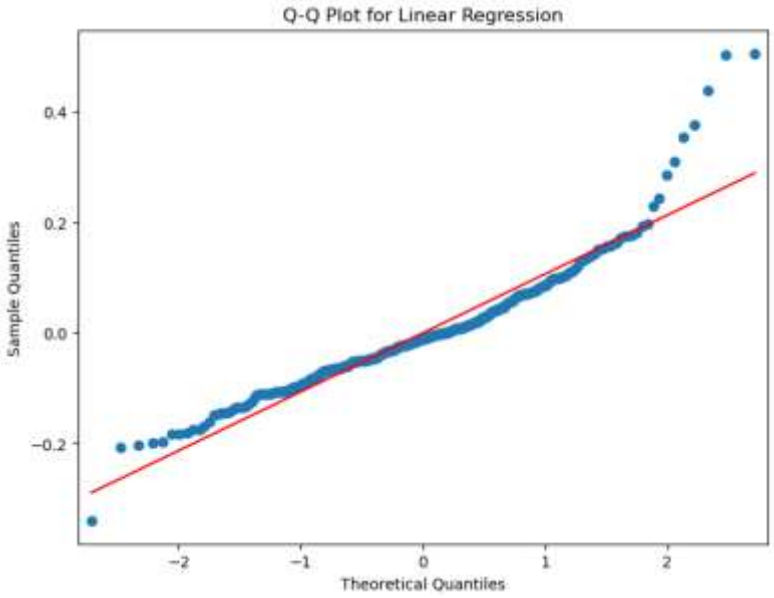

Figure 3: Q-Q Plot for Linear Regression

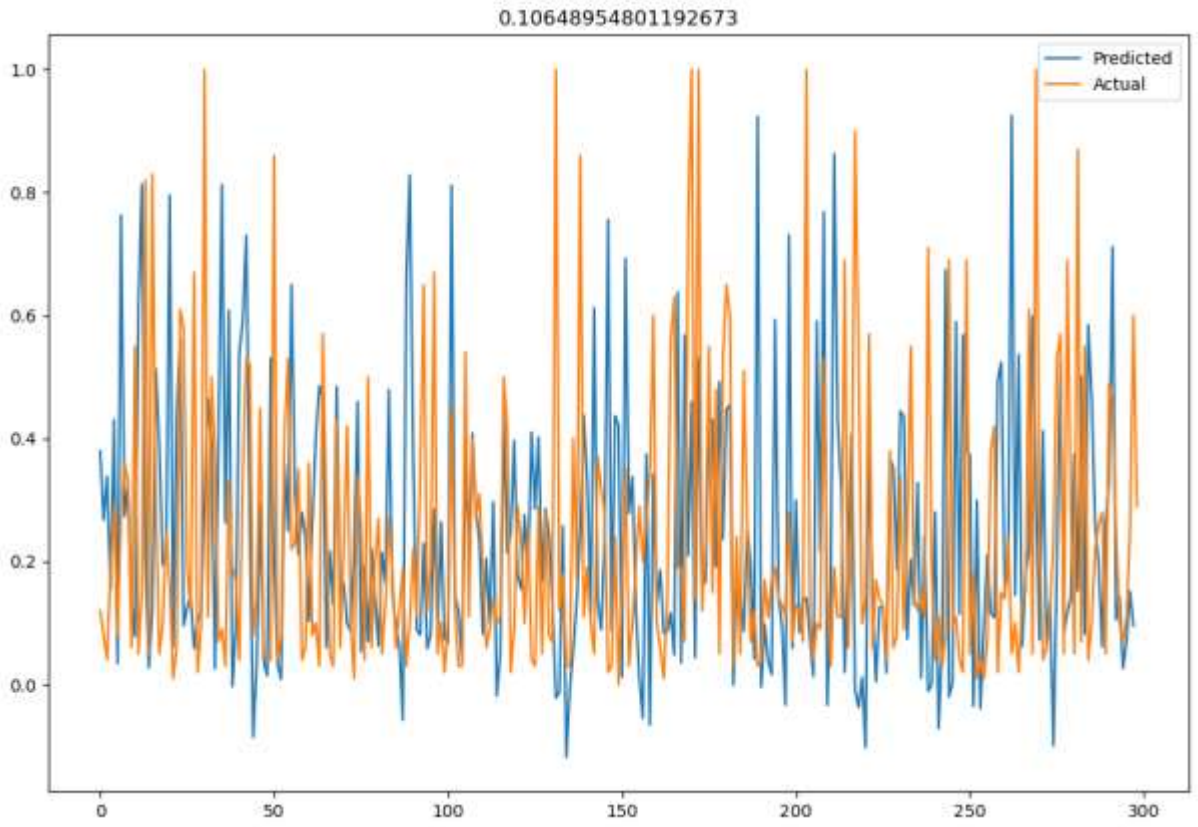

Figure 4: Running model on test data

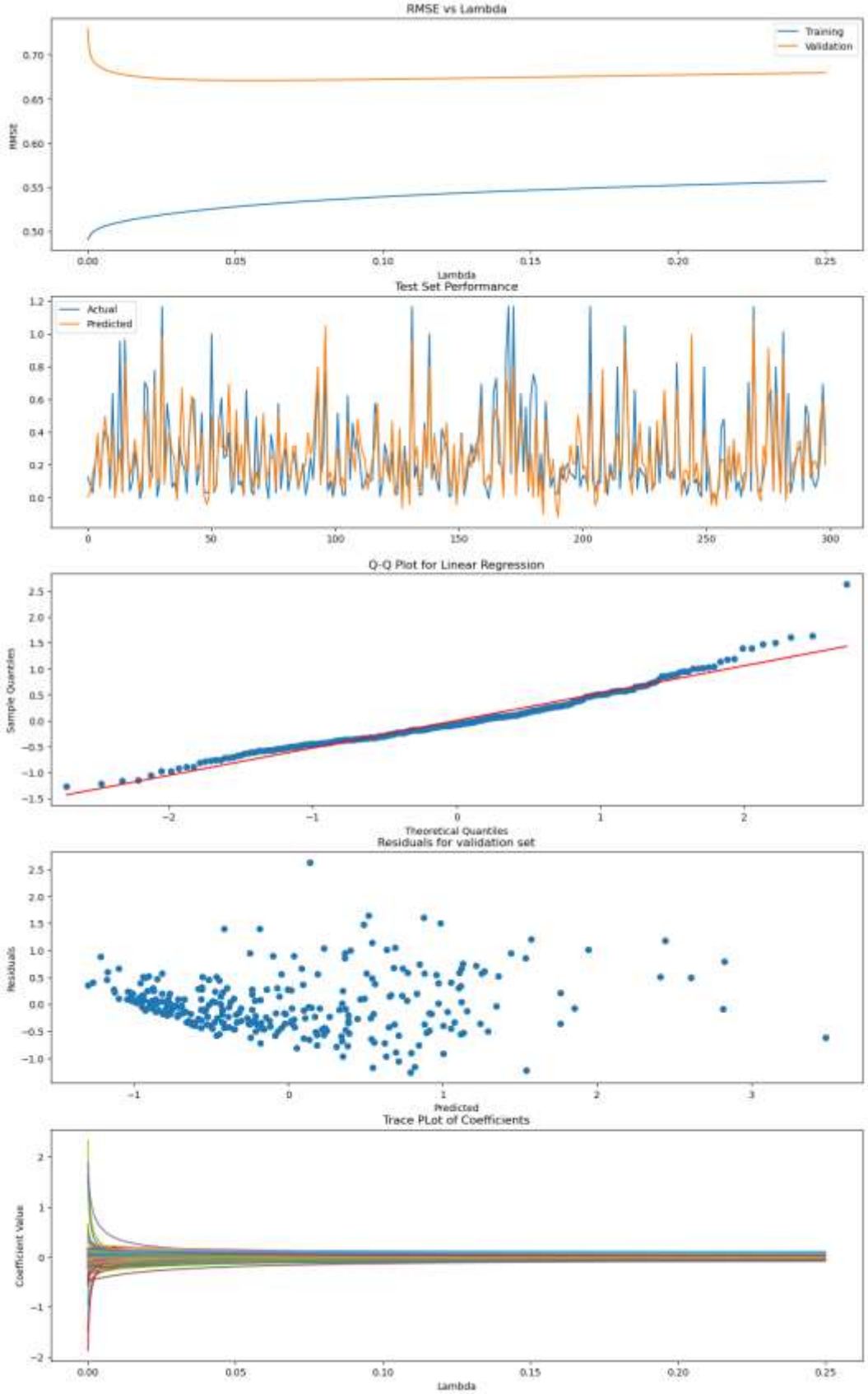

Figure 5: Ridge Regression results

```
Best values on Validation Data set
Best R Squared = 0.7197807202217746
Best Adjusted = 0.5775374309942491
Best RMSE (val) = 0.16885489509642823
Best RMSE (test) = 0.15612621150877118
```

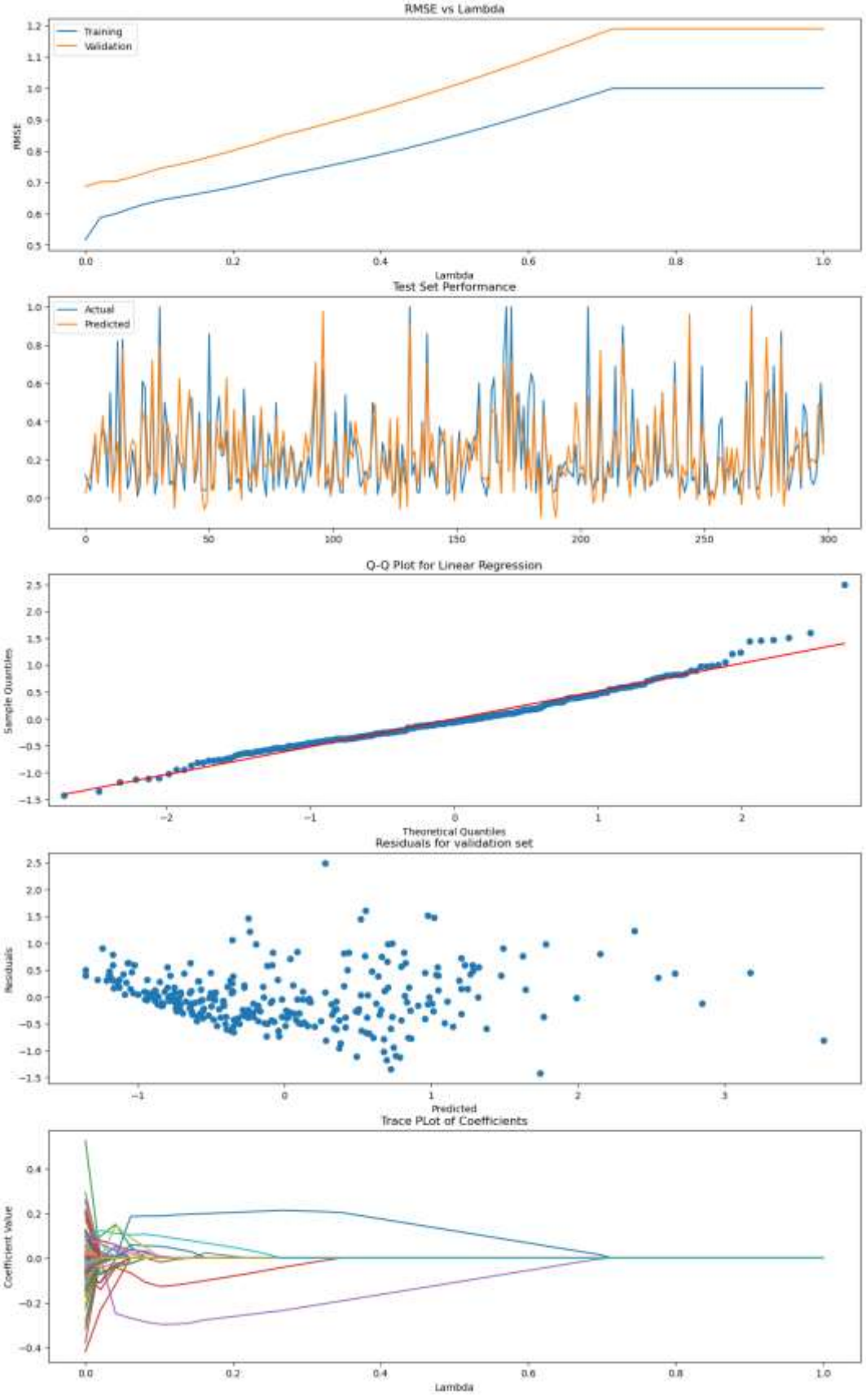

Figure 6: Lasso Regression results

```
Best values on Validation Data set
Best R Squared = 0.7320517436259424
Best Adjusted = 0.5960374002888573
Best RMSE (val) = 0.14582934578499862
Best RMSE (test) = 0.14050173286050835
```

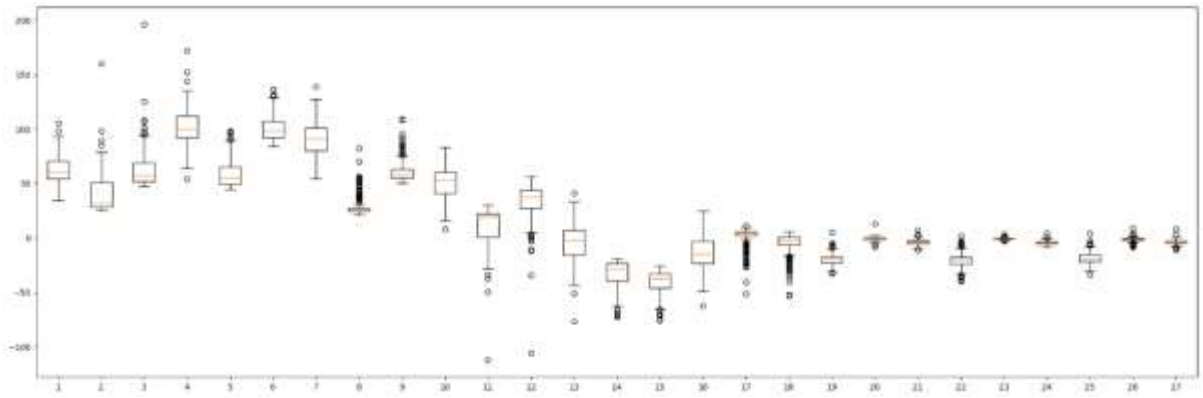

Figure 7: Raw Data Boxplot

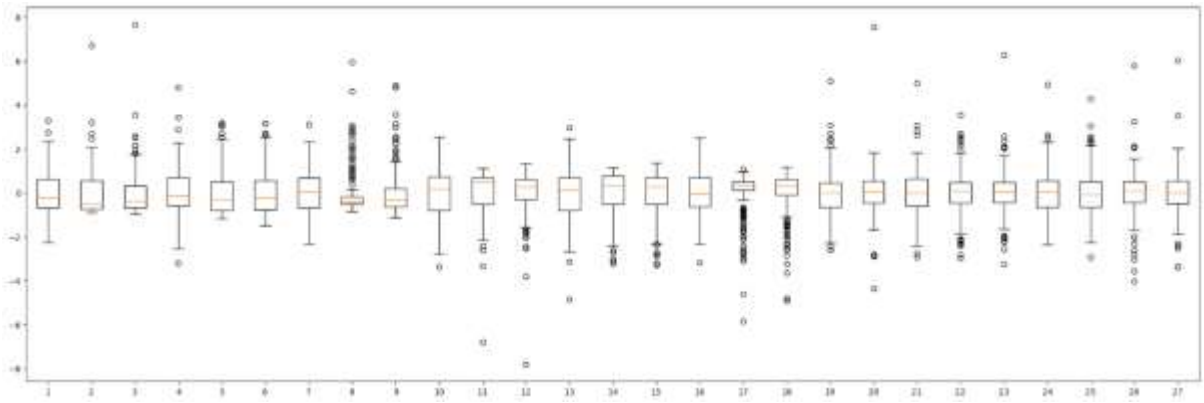

Figure 8: Standardised Data Boxplot

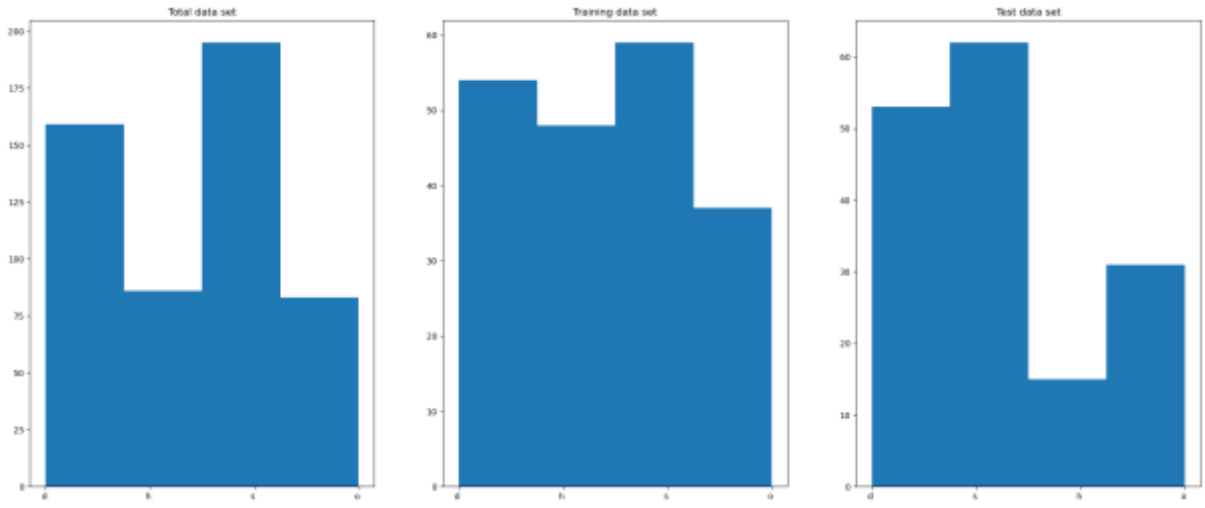

Figure 9: Some class imbalance

```
knn = KNeighborsClassifier()
k_range = list(range(1, 64))
param_grid = dict(n_neighbors=k_range)

# defining parameter range
grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', return_train_score=False,verbose=1)

# fitting the model for grid search
grid_search=grid.fit(X_val, Y_val)

print(grid_search.best_params_)
```

```
Fitting 10 folds for each of 63 candidates, totalling 630 fits
{'n_neighbors': 3}
```

Figure 10: CKNN Grid Search

Figure 11: n_neighbors = 3 CKNN model evaluated over the testing set

```
rf = RandomForestClassifier(random_state=0)
param_grid = {
    'n_estimators': [25, 50, 100, 200],
    'max_depth' : [1,2,3],
    'class_weight' :['balanced', 'balanced_subsample', None]
}
CV_rf = GridSearchCV(estimator=rf, param_grid=param_grid, cv= 5)
grid_search = CV_rf.fit(X_val, Y_val)
print(grid_search.best_params_)
```

```
{'class_weight': 'balanced', 'max_depth': 3, 'n_estimators': 25}
```

Figure 12: Random Forest Grid Search

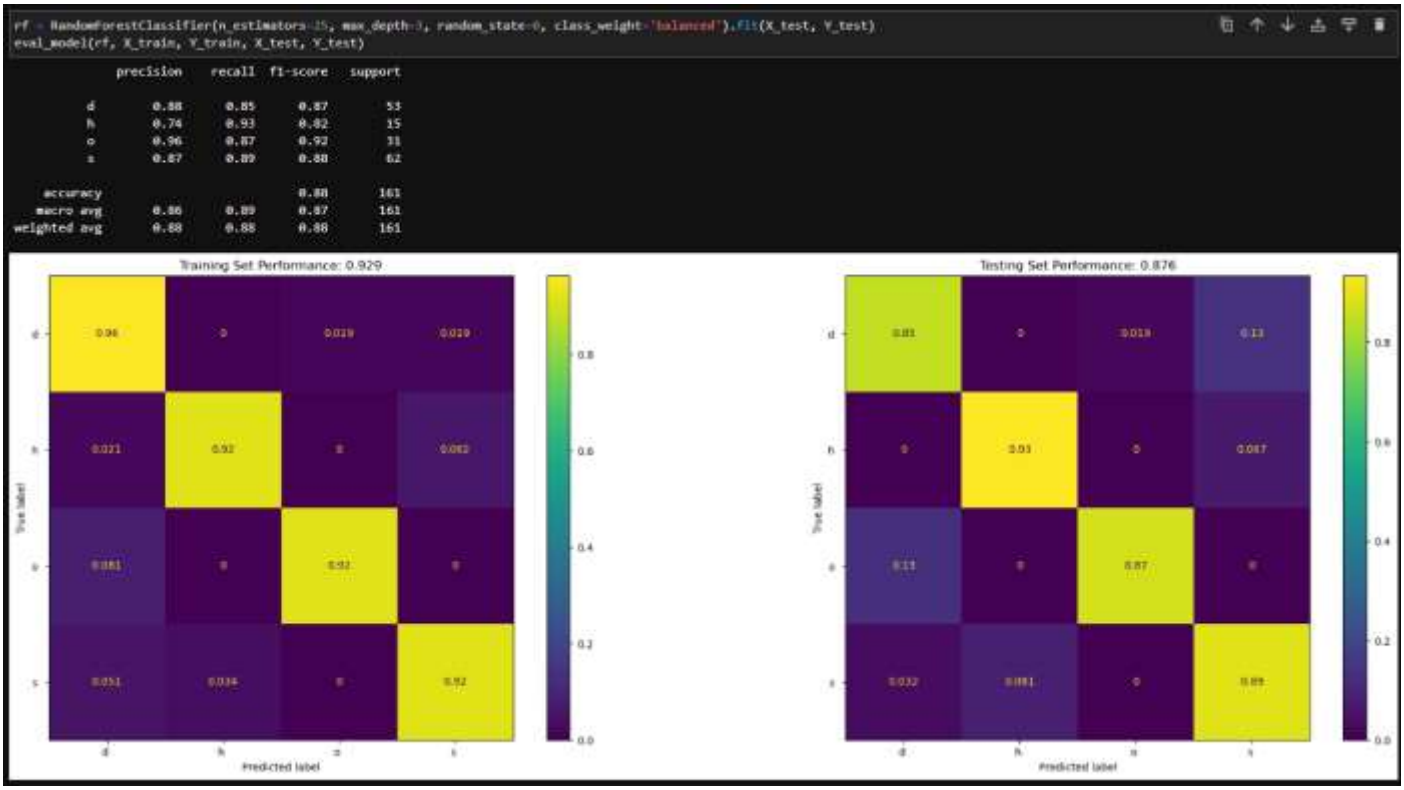

Figure 13: Random Forest evaluated on testing set based on grid search

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel' : ['linear', 'rbf', 'poly'],
    'gamma' :[0.1, 0.01, 0.001],
    'degree' : [4, 5, 6, 7, 8]
}
svm = SVC(class_weight='balanced')
grid_search = GridSearchCV(svm, param_grid)
grid_search.fit(X_val, Y_val)
```

Figure 14: SVM grid search code

```
best_system = np.argmin(grid_search.cv_results_['rank_test_score'])
params = grid_search.cv_results_['params'][best_system]
print(params)
svm = SVC().set_params(**params)
svm.fit(X_test, Y_test)
eval_model(svm, X_val, Y_val, X_test, Y_test)
```

```
{'C': 0.1, 'degree': 6, 'gamma': 0.1, 'kernel': 'poly'}
              precision    recall  f1-score   support

           d       1.00      0.40      0.57        53
           h       1.00      0.33      0.50        15
           o       1.00      0.81      0.89        31
           s       0.56      1.00      0.72        62

    accuracy                           0.70       161
   macro avg       0.89      0.63      0.67       161
weighted avg       0.83      0.70      0.68       161
```

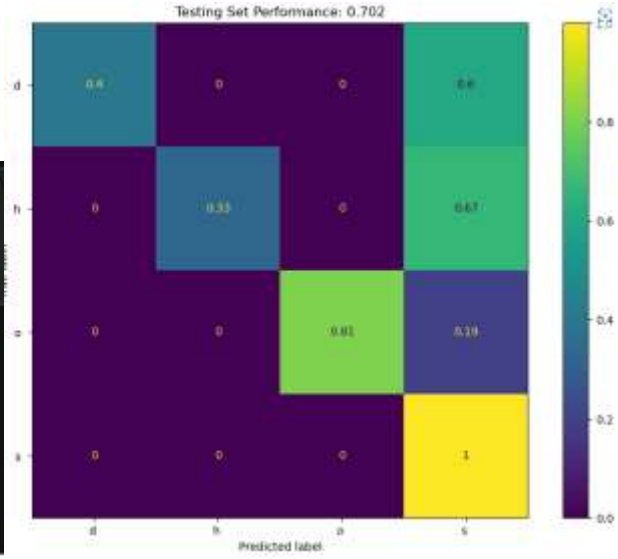

Figure 15: SVM testing set evaluation

Figure 16: Non-augmented data

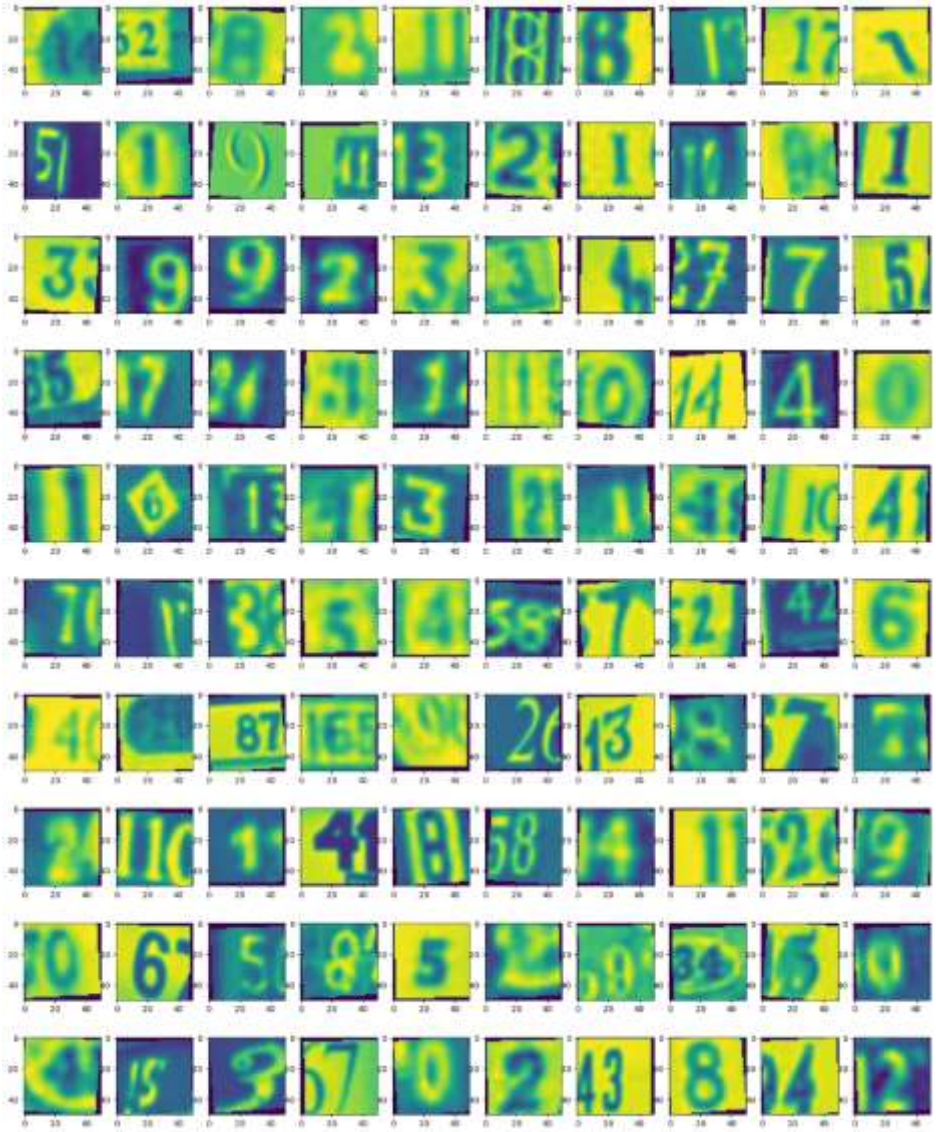

Figure 17: Augmented data

```
Model: "vgg_for_svhn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 8)         224

 activation (Activation)     (None, 32, 32, 8)         0

 conv2d_1 (Conv2D)           (None, 32, 32, 8)         584

 activation_1 (Activation)   (None, 32, 32, 8)         0

 max_pooling2d (MaxPooling2D  (None, 16, 16, 8)        0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 16)        1168

 activation_2 (Activation)   (None, 16, 16, 16)        0

 conv2d_3 (Conv2D)           (None, 16, 16, 16)        2320

 activation_3 (Activation)   (None, 16, 16, 16)        0

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 16)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          4640

 activation_4 (Activation)   (None, 8, 8, 32)          0

 conv2d_5 (Conv2D)           (None, 8, 8, 32)          9248

 activation_5 (Activation)   (None, 8, 8, 32)          0

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 285,970
Trainable params: 285,970
Non-trainable params: 0
```

Figure 18: VGG-like network for non-augmented data

```
Model: "vgg_for_svhn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 augmented (InputLayer)      [(None, 50, 50, 1)]       0

 conv2d_6 (Conv2D)           (None, 50, 50, 8)         80

 activation_6 (Activation)   (None, 50, 50, 8)         0

 conv2d_7 (Conv2D)           (None, 50, 50, 8)         584

 activation_7 (Activation)   (None, 50, 50, 8)         0

 max_pooling2d_3 (MaxPooling  (None, 25, 25, 8)        0
 2D)

 conv2d_8 (Conv2D)           (None, 25, 25, 16)        1168

 activation_8 (Activation)   (None, 25, 25, 16)        0

 conv2d_9 (Conv2D)           (None, 25, 25, 16)        2320

 activation_9 (Activation)   (None, 25, 25, 16)        0

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 16)       0
 2D)

 conv2d_10 (Conv2D)          (None, 12, 12, 32)        4640

 activation_10 (Activation)  (None, 12, 12, 32)        0

 conv2d_11 (Conv2D)          (None, 12, 12, 32)        9248

 activation_11 (Activation)  (None, 12, 12, 32)        0

 max_pooling2d_5 (MaxPooling  (None, 6, 6, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 1152)              0

 dense_2 (Dense)             (None, 512)               590336

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 613,506
Trainable params: 613,506
Non-trainable params: 0
```

Figure 19: VGG-like network for augmented data

```
Epoch 1/32
8/8 [==============================] - 4s 285ms/step - loss: 2.2722 - accuracy: 0.1460 - val_loss: 2.2409 - val_accuracy: 0.1894
Epoch 2/32
8/8 [==============================] - 2s 253ms/step - loss: 2.2510 - accuracy: 0.1750 - val_loss: 2.2380 - val_accuracy: 0.1894
Epoch 3/32
8/8 [==============================] - 2s 266ms/step - loss: 2.2419 - accuracy: 0.1790 - val_loss: 2.2346 - val_accuracy: 0.1910
Epoch 4/32
8/8 [==============================] - 2s 269ms/step - loss: 2.2379 - accuracy: 0.1760 - val_loss: 2.2287 - val_accuracy: 0.1894
Epoch 5/32
8/8 [==============================] - 2s 257ms/step - loss: 2.2259 - accuracy: 0.1750 - val_loss: 2.2158 - val_accuracy: 0.2117
Epoch 6/32
8/8 [==============================] - 2s 257ms/step - loss: 2.2017 - accuracy: 0.2100 - val_loss: 2.1824 - val_accuracy: 0.1925
Epoch 7/32
8/8 [==============================] - 2s 255ms/step - loss: 2.1279 - accuracy: 0.2300 - val_loss: 2.1249 - val_accuracy: 0.2228
Epoch 8/32
8/8 [==============================] - 2s 250ms/step - loss: 2.0083 - accuracy: 0.2940 - val_loss: 2.0147 - val_accuracy: 0.2872
Epoch 9/32
8/8 [==============================] - 2s 255ms/step - loss: 1.8676 - accuracy: 0.3490 - val_loss: 1.8647 - val_accuracy: 0.3603
Epoch 10/32
8/8 [==============================] - 2s 265ms/step - loss: 1.6726 - accuracy: 0.4390 - val_loss: 1.7469 - val_accuracy: 0.4156
Epoch 11/32
8/8 [==============================] - 2s 254ms/step - loss: 1.4960 - accuracy: 0.4980 - val_loss: 1.6140 - val_accuracy: 0.4580
Epoch 12/32
8/8 [==============================] - 2s 252ms/step - loss: 1.3513 - accuracy: 0.5670 - val_loss: 1.5449 - val_accuracy: 0.5129
Epoch 13/32
8/8 [==============================] - 2s 246ms/step - loss: 1.2194 - accuracy: 0.6030 - val_loss: 1.4453 - val_accuracy: 0.5364
Epoch 14/32
8/8 [==============================] - 2s 247ms/step - loss: 1.0326 - accuracy: 0.6580 - val_loss: 1.3796 - val_accuracy: 0.5534
Epoch 15/32
8/8 [==============================] - 2s 251ms/step - loss: 0.8963 - accuracy: 0.7130 - val_loss: 1.3737 - val_accuracy: 0.6001
Epoch 16/32
8/8 [==============================] - 2s 249ms/step - loss: 0.8046 - accuracy: 0.7360 - val_loss: 1.2848 - val_accuracy: 0.6127
Epoch 17/32
8/8 [==============================] - 2s 247ms/step - loss: 0.7079 - accuracy: 0.7780 - val_loss: 1.3242 - val_accuracy: 0.6178
Epoch 18/32
8/8 [==============================] - 2s 246ms/step - loss: 0.5886 - accuracy: 0.8180 - val_loss: 1.3228 - val_accuracy: 0.6329
Epoch 19/32
8/8 [==============================] - 2s 243ms/step - loss: 0.4917 - accuracy: 0.8510 - val_loss: 1.4397 - val_accuracy: 0.6239
Epoch 20/32
8/8 [==============================] - 2s 248ms/step - loss: 0.4987 - accuracy: 0.8320 - val_loss: 1.3786 - val_accuracy: 0.6370
Epoch 21/32
8/8 [==============================] - 2s 251ms/step - loss: 0.3602 - accuracy: 0.8910 - val_loss: 1.3743 - val_accuracy: 0.6639
Epoch 22/32
8/8 [==============================] - 2s 250ms/step - loss: 0.2716 - accuracy: 0.9180 - val_loss: 1.4480 - val_accuracy: 0.6716
Epoch 23/32
8/8 [==============================] - 2s 254ms/step - loss: 0.2237 - accuracy: 0.9360 - val_loss: 1.5339 - val_accuracy: 0.6726
Epoch 24/32
8/8 [==============================] - 2s 254ms/step - loss: 0.1710 - accuracy: 0.9590 - val_loss: 1.5681 - val_accuracy: 0.6805
Epoch 25/32
8/8 [==============================] - 2s 256ms/step - loss: 0.1339 - accuracy: 0.9630 - val_loss: 1.7021 - val_accuracy: 0.6977
Epoch 26/32
8/8 [==============================] - 2s 256ms/step - loss: 0.0916 - accuracy: 0.9810 - val_loss: 1.7015 - val_accuracy: 0.7022
Epoch 27/32
8/8 [==============================] - 2s 257ms/step - loss: 0.0864 - accuracy: 0.9790 - val_loss: 1.9223 - val_accuracy: 0.6941
Epoch 28/32
8/8 [==============================] - 2s 259ms/step - loss: 0.0625 - accuracy: 0.9870 - val_loss: 1.9369 - val_accuracy: 0.6996
Epoch 29/32
8/8 [==============================] - 2s 256ms/step - loss: 0.0408 - accuracy: 0.9920 - val_loss: 1.9888 - val_accuracy: 0.7015
Epoch 30/32
8/8 [==============================] - 2s 260ms/step - loss: 0.0326 - accuracy: 0.9950 - val_loss: 2.1415 - val_accuracy: 0.6984
Epoch 31/32
8/8 [==============================] - 2s 256ms/step - loss: 0.0332 - accuracy: 0.9930 - val_loss: 2.3010 - val_accuracy: 0.7101
Epoch 32/32
8/8 [==============================] - 2s 258ms/step - loss: 0.0389 - accuracy: 0.9880 - val_loss: 2.2574 - val_accuracy: 0.6932
```

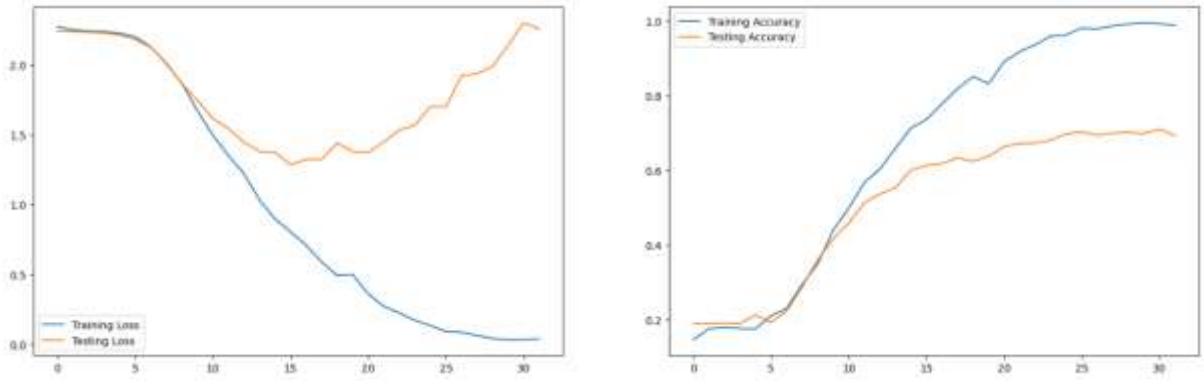

Figure 20: Non-augmented results

```
Training Time: 0.015625
Inference Time: 35.656250
```

```
Epoch 1/64
8/8 [==============================] - 5s 536ms/step - loss: 2.2735 - accuracy: 0.1520 - val_loss: 2.2403 - val_accuracy: 0.1894
Epoch 2/64
8/8 [==============================] - 4s 481ms/step - loss: 2.2514 - accuracy: 0.1750 - val_loss: 2.2419 - val_accuracy: 0.1894
Epoch 3/64
8/8 [==============================] - 4s 479ms/step - loss: 2.2449 - accuracy: 0.1750 - val_loss: 2.2375 - val_accuracy: 0.1894
Epoch 4/64
8/8 [==============================] - 3s 465ms/step - loss: 2.2429 - accuracy: 0.1750 - val_loss: 2.2369 - val_accuracy: 0.1894
Epoch 5/64
8/8 [==============================] - 3s 476ms/step - loss: 2.2420 - accuracy: 0.1750 - val_loss: 2.2358 - val_accuracy: 0.1894
Epoch 6/64
8/8 [==============================] - 4s 484ms/step - loss: 2.2430 - accuracy: 0.1750 - val_loss: 2.2351 - val_accuracy: 0.1894
Epoch 7/64
8/8 [==============================] - 3s 465ms/step - loss: 2.2395 - accuracy: 0.1750 - val_loss: 2.2356 - val_accuracy: 0.1894
Epoch 8/64
8/8 [==============================] - 4s 476ms/step - loss: 2.2379 - accuracy: 0.1750 - val_loss: 2.2331 - val_accuracy: 0.1894
Epoch 9/64
8/8 [==============================] - 3s 457ms/step - loss: 2.2342 - accuracy: 0.1750 - val_loss: 2.2286 - val_accuracy: 0.1894
Epoch 10/64
8/8 [==============================] - 3s 453ms/step - loss: 2.2275 - accuracy: 0.1770 - val_loss: 2.2205 - val_accuracy: 0.1903
Epoch 11/64
8/8 [==============================] - 3s 452ms/step - loss: 2.2139 - accuracy: 0.1970 - val_loss: 2.2033 - val_accuracy: 0.2082
Epoch 12/64
8/8 [==============================] - 3s 446ms/step - loss: 2.1800 - accuracy: 0.2210 - val_loss: 2.1845 - val_accuracy: 0.2105
Epoch 13/64
8/8 [==============================] - 4s 480ms/step - loss: 2.1447 - accuracy: 0.2360 - val_loss: 2.1240 - val_accuracy: 0.2316
Epoch 14/64
8/8 [==============================] - 3s 473ms/step - loss: 2.0791 - accuracy: 0.2540 - val_loss: 2.0464 - val_accuracy: 0.2818
Epoch 15/64
8/8 [==============================] - 3s 451ms/step - loss: 1.9700 - accuracy: 0.3040 - val_loss: 1.9365 - val_accuracy: 0.3093
Epoch 16/64
8/8 [==============================] - 3s 448ms/step - loss: 1.8104 - accuracy: 0.3940 - val_loss: 1.7572 - val_accuracy: 0.4026
Epoch 17/64
8/8 [==============================] - 3s 467ms/step - loss: 1.6392 - accuracy: 0.4580 - val_loss: 1.6320 - val_accuracy: 0.4497
Epoch 18/64
8/8 [==============================] - 3s 472ms/step - loss: 1.4843 - accuracy: 0.5240 - val_loss: 1.4789 - val_accuracy: 0.5225
Epoch 19/64
8/8 [==============================] - 3s 459ms/step - loss: 1.3154 - accuracy: 0.5740 - val_loss: 1.3780 - val_accuracy: 0.5545
Epoch 20/64
8/8 [==============================] - 3s 468ms/step - loss: 1.1645 - accuracy: 0.6370 - val_loss: 1.2573 - val_accuracy: 0.5986
Epoch 21/64
8/8 [==============================] - 3s 454ms/step - loss: 1.0727 - accuracy: 0.6550 - val_loss: 1.1817 - val_accuracy: 0.6308
Epoch 22/64
8/8 [==============================] - 3s 473ms/step - loss: 0.9364 - accuracy: 0.6900 - val_loss: 1.1978 - val_accuracy: 0.6301
Epoch 23/64
8/8 [==============================] - 3s 468ms/step - loss: 0.8275 - accuracy: 0.7460 - val_loss: 1.1263 - val_accuracy: 0.6722
Epoch 24/64
8/8 [==============================] - 3s 467ms/step - loss: 0.7912 - accuracy: 0.7440 - val_loss: 1.0820 - val_accuracy: 0.6814
Epoch 25/64
8/8 [==============================] - 3s 458ms/step - loss: 0.7014 - accuracy: 0.7800 - val_loss: 1.0081 - val_accuracy: 0.6992
Epoch 26/64
8/8 [==============================] - 3s 454ms/step - loss: 0.6287 - accuracy: 0.7980 - val_loss: 1.0783 - val_accuracy: 0.6932
Epoch 27/64
8/8 [==============================] - 4s 479ms/step - loss: 0.6083 - accuracy: 0.8070 - val_loss: 1.0039 - val_accuracy: 0.7095
Epoch 28/64
8/8 [==============================] - 4s 503ms/step - loss: 0.5462 - accuracy: 0.8280 - val_loss: 0.9697 - val_accuracy: 0.7183
Epoch 29/64
8/8 [==============================] - 3s 473ms/step - loss: 0.4893 - accuracy: 0.8550 - val_loss: 0.9629 - val_accuracy: 0.7378
Epoch 30/64
8/8 [==============================] - 4s 503ms/step - loss: 0.4445 - accuracy: 0.8530 - val_loss: 0.9658 - val_accuracy: 0.7422
Epoch 31/64
8/8 [==============================] - 4s 502ms/step - loss: 0.3338 - accuracy: 0.8990 - val_loss: 0.9134 - val_accuracy: 0.7634
Epoch 32/64
8/8 [==============================] - 3s 463ms/step - loss: 0.2939 - accuracy: 0.9070 - val_loss: 1.0678 - val_accuracy: 0.7512
Epoch 33/64
8/8 [==============================] - 3s 475ms/step - loss: 0.3093 - accuracy: 0.9120 - val_loss: 1.0707 - val_accuracy: 0.7507
Epoch 34/64
8/8 [==============================] - 3s 474ms/step - loss: 0.2631 - accuracy: 0.9160 - val_loss: 1.0396 - val_accuracy: 0.7427
```

```
Epoch 35/64
8/8 [==============================] - 3s 460ms/step - loss: 0.2616 - accuracy: 0.9190 - val_loss: 1.0079 - val_accuracy: 0.7622
Epoch 36/64
8/8 [==============================] - 4s 476ms/step - loss: 0.1921 - accuracy: 0.9400 - val_loss: 1.0569 - val_accuracy: 0.7667
Epoch 37/64
8/8 [==============================] - 3s 464ms/step - loss: 0.1967 - accuracy: 0.9430 - val_loss: 1.1657 - val_accuracy: 0.7583
Epoch 38/64
8/8 [==============================] - 4s 494ms/step - loss: 0.1629 - accuracy: 0.9450 - val_loss: 1.1393 - val_accuracy: 0.7685
Epoch 39/64
8/8 [==============================] - 4s 496ms/step - loss: 0.1550 - accuracy: 0.9460 - val_loss: 1.1432 - val_accuracy: 0.7766
Epoch 40/64
8/8 [==============================] - 4s 489ms/step - loss: 0.1576 - accuracy: 0.9580 - val_loss: 1.1610 - val_accuracy: 0.7719
Epoch 41/64
8/8 [==============================] - 3s 468ms/step - loss: 0.1454 - accuracy: 0.9530 - val_loss: 1.2078 - val_accuracy: 0.7551
Epoch 42/64
8/8 [==============================] - 3s 446ms/step - loss: 0.1354 - accuracy: 0.9560 - val_loss: 1.1658 - val_accuracy: 0.7731
Epoch 43/64
8/8 [==============================] - 4s 501ms/step - loss: 0.1326 - accuracy: 0.9590 - val_loss: 1.2136 - val_accuracy: 0.7612
Epoch 44/64
8/8 [==============================] - 3s 469ms/step - loss: 0.0858 - accuracy: 0.9770 - val_loss: 1.2205 - val_accuracy: 0.7759
Epoch 45/64
8/8 [==============================] - 3s 474ms/step - loss: 0.0858 - accuracy: 0.9770 - val_loss: 1.1636 - val_accuracy: 0.7731
Epoch 46/64
8/8 [==============================] - 4s 484ms/step - loss: 0.0710 - accuracy: 0.9850 - val_loss: 1.2532 - val_accuracy: 0.7749
Epoch 47/64
8/8 [==============================] - 4s 539ms/step - loss: 0.0777 - accuracy: 0.9780 - val_loss: 1.2815 - val_accuracy: 0.7739
Epoch 48/64
8/8 [==============================] - 4s 481ms/step - loss: 0.0850 - accuracy: 0.9700 - val_loss: 1.3804 - val_accuracy: 0.7636
Epoch 49/64
8/8 [==============================] - 3s 456ms/step - loss: 0.0456 - accuracy: 0.9870 - val_loss: 1.4944 - val_accuracy: 0.7648
Epoch 50/64
8/8 [==============================] - 3s 456ms/step - loss: 0.0695 - accuracy: 0.9800 - val_loss: 1.4694 - val_accuracy: 0.7692
Epoch 51/64
8/8 [==============================] - 3s 460ms/step - loss: 0.0625 - accuracy: 0.9800 - val_loss: 1.4640 - val_accuracy: 0.7694
Epoch 52/64
8/8 [==============================] - 4s 480ms/step - loss: 0.0621 - accuracy: 0.9750 - val_loss: 1.4316 - val_accuracy: 0.7669
Epoch 53/64
8/8 [==============================] - 4s 504ms/step - loss: 0.0443 - accuracy: 0.9870 - val_loss: 1.4321 - val_accuracy: 0.7733
Epoch 54/64
8/8 [==============================] - 3s 456ms/step - loss: 0.0593 - accuracy: 0.9840 - val_loss: 1.4530 - val_accuracy: 0.7743
Epoch 55/64
8/8 [==============================] - 3s 450ms/step - loss: 0.0671 - accuracy: 0.9770 - val_loss: 1.5424 - val_accuracy: 0.7732
Epoch 56/64
8/8 [==============================] - 3s 453ms/step - loss: 0.0596 - accuracy: 0.9830 - val_loss: 1.4204 - val_accuracy: 0.7741
Epoch 57/64
8/8 [==============================] - 3s 450ms/step - loss: 0.0984 - accuracy: 0.9770 - val_loss: 1.3263 - val_accuracy: 0.7739
Epoch 58/64
8/8 [==============================] - 3s 450ms/step - loss: 0.0526 - accuracy: 0.9850 - val_loss: 1.3790 - val_accuracy: 0.7719
Epoch 59/64
8/8 [==============================] - 3s 447ms/step - loss: 0.0585 - accuracy: 0.9830 - val_loss: 1.3597 - val_accuracy: 0.7653
Epoch 60/64
8/8 [==============================] - 3s 451ms/step - loss: 0.0698 - accuracy: 0.9820 - val_loss: 1.3746 - val_accuracy: 0.7854
Epoch 61/64
8/8 [==============================] - 3s 449ms/step - loss: 0.0541 - accuracy: 0.9770 - val_loss: 1.3226 - val_accuracy: 0.7782
Epoch 62/64
8/8 [==============================] - 3s 456ms/step - loss: 0.0476 - accuracy: 0.9840 - val_loss: 1.3059 - val_accuracy: 0.7812
Epoch 63/64
8/8 [==============================] - 3s 447ms/step - loss: 0.0413 - accuracy: 0.9940 - val_loss: 1.4461 - val_accuracy: 0.7836
Epoch 64/64
8/8 [==============================] - 3s 456ms/step - loss: 0.0395 - accuracy: 0.9890 - val_loss: 1.4754 - val_accuracy: 0.7743
```

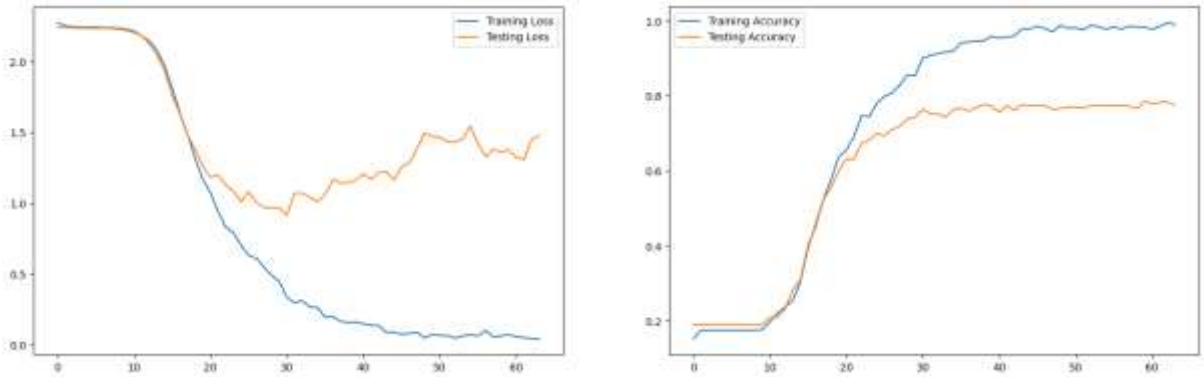

Figure 21: Augmented results

```
Training Time: 0.015625
Inference Time: 157.171875
```

```
Training Time: 1.343750
Inference Time (training set): 0.250000
Inference Time (testing set): 3.937500
              precision    recall  f1-score   support

           0       0.25      0.23      0.24       711
           1       0.33      0.51      0.40      1894
           2       0.33      0.37      0.35      1497
           3       0.31      0.25      0.28      1141
           4       0.28      0.27      0.27      1035
           5       0.33      0.29      0.31       892
           6       0.28      0.17      0.21       758
           7       0.31      0.21      0.25       789
           8       0.30      0.26      0.28       683
           9       0.41      0.25      0.31       600

    accuracy                           0.31     10000
   macro avg       0.31      0.28      0.29     10000
weighted avg       0.31      0.31      0.31     10000
```

Figure 22: SVM results

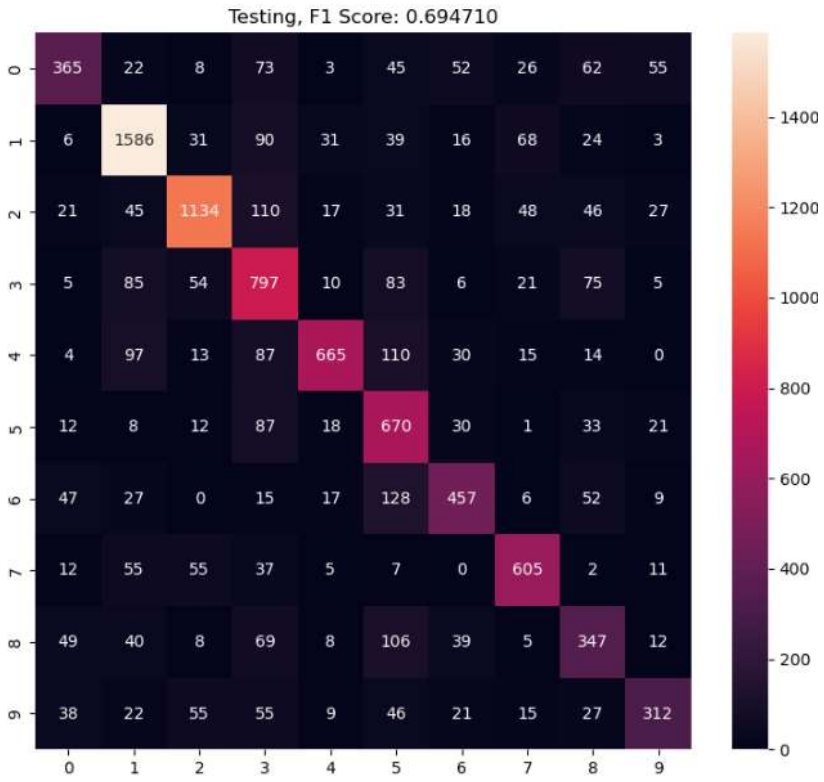

Figure 23: Testing set F1 scores and confusion chart for non-augmented data

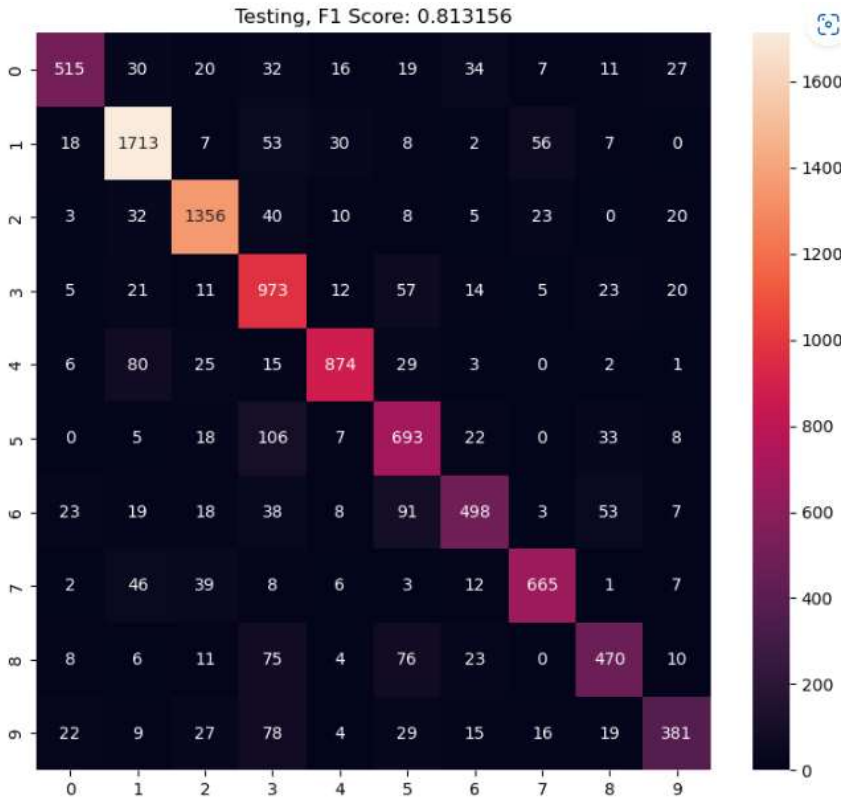

Figure 24: Testing set F1 scores and confusion chart for augmented data

References

- Jaadi, Z. (2019, September 4). When and why to standardize your data. Built In. https://builtin.com/data-science/when-and-why-standardize-your-data
- Pluviophile. (2022, December 8). Which standardization technique to use for lasso regression? Data Science Stack Exchange. https://datascience.stackexchange.com/questions/116858/which-standardization-technique-to-use-for-lasso-regression#:~:text=It%20is%20generally%20recommended%20to,of%20the%20Lasso%20regression%20model.
- Pedregosa, et al. (2011) Scikit-learn: Machine Learning in Python., *JMLR 12*, 2825-2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
- Khoong, W. H. (2021, August 29). When do support Vector Machines Fail? Medium. https://towardsdatascience.com/when-do-support-vector-machines-fail-3f23295ebef2