CS-UY 4563: Machine Learning
Professor Linda Sellie
Section B
Final Project Report
Loan Grade Classification for Credit Risk
December 7, 2022
Shubh Savani, Sunan Tajwar

1. Introduction

For this final project, we used a dataset from OpenML that contained multiple attributes of data banks and financial institutions keep track of when dealing with and tracking distributed loans, in order to determine the degree of credit risk. The dataset contains 12 features and 32581 instances. One of these instances is *loan grade*, which is essentially a letter score assigned to a loan based on a borrower's credit history, collateral, and the likelihood of repayment based on their previous history. In our model, we will try and predict the loan grade based on the other features in the model. The other original features included in the data set are person_age, person_income, person_home_ownership, person_employment_length, loan_intent, loan_grade, loan_amount, loan_interest_rate, loan_status, loan_percent_income, default_on_file, and and credit_history_length.

We have attempted to build this model using a multiclass classification problem, with the goal of predicting the loan grade from each instance, with the values ranging from 'A' to 'G', based on the historical data in the other features. In order to do so we have attempted using different variations of three types of classification models: logistic regression, support-vector-machines (SVM), and neural networks.

2. Data Preprocessing / Preparation

2.1 Data Cleaning

When observing and researching the twelve different features included in the original dataset, we concluded that all eleven predictive features were important to determining the loan_grade. We came to the same conclusion when producing a correlation matrix/heatmap between the features, in which we were unable to distinguish any features that were non-significant. Thus we initially chose to not remove any features in our initial run-through of the models. There were also multiple instances in which one or more of the features of the instance did not have a set numeric or categorical value, and in its place stored a '?'. To account for this, we replaced all '?' values in the dataset with 'NaN' values. From there, because we had a large enough dataset of size 32581 to both train and test on, we decided to simply drop all instances which included a 'NaN' value. In doing so we were still left with around 28000 instances in the dataset, and so believed it would not be necessary to impute data for 'NaN' values. We were unable to distinguish any outliers in the data, and so we felt after cleaning out the dataset by dropping instances with 'NaN' values, we could proceed with the data preprocessing.

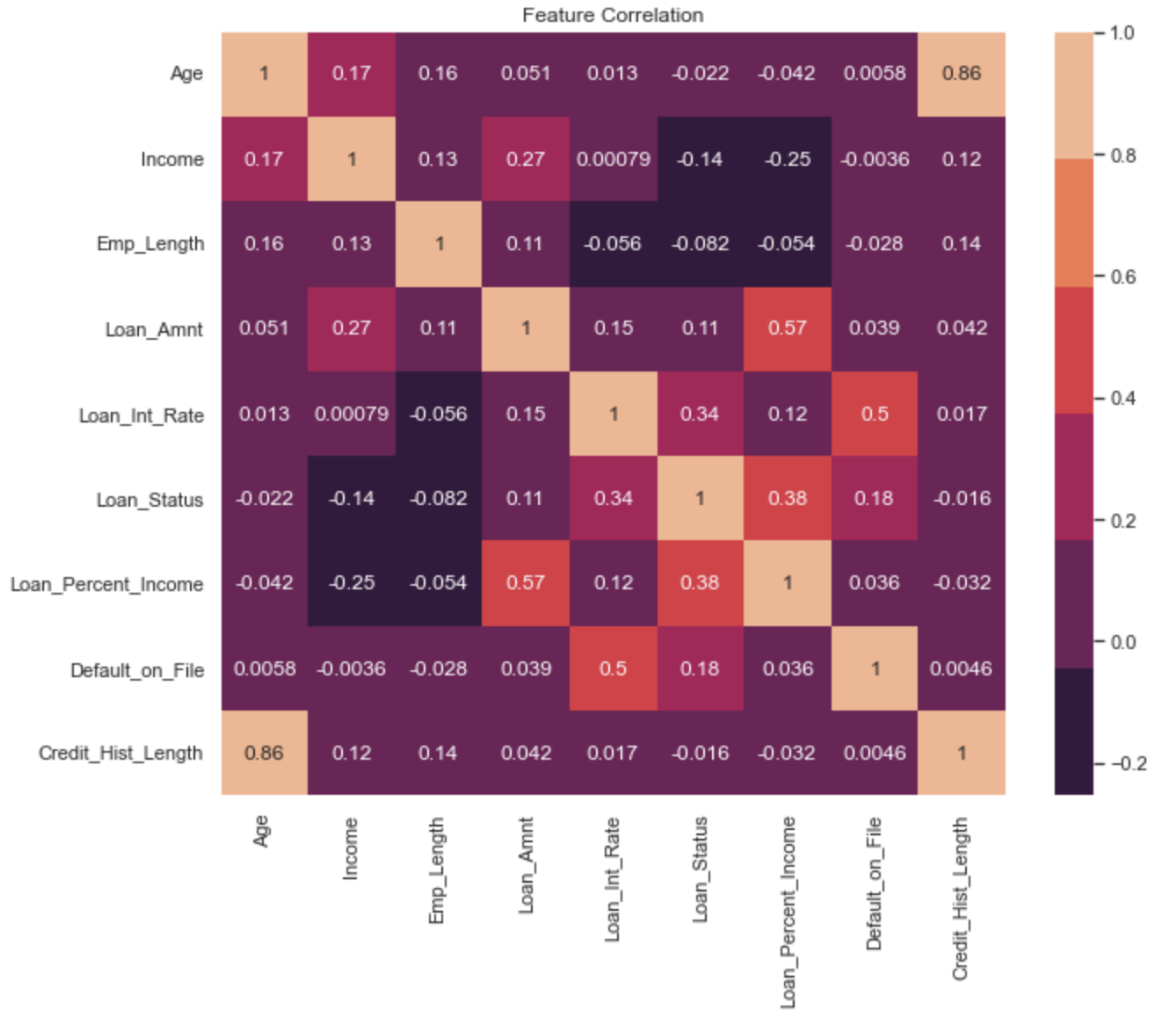

Figure 1: Features Heatmap

2.2 Categorial Encoding

To start, for the credit_default_on_file feature, the data was initially either 'Y' or 'N'. As all of the other binary categorical features were of data '1' or '0', for credit_default_on_file, we replaced 'Y' with '1' and 'N' with '0'. We then split our prediction features into two matrices, *X_num* and *X_cat*. X_num contained all of our already numerical features, which included: age, income, employment_length, loan_amount, loan_interest_rate, and loan_status. X_cat contained the categorical features home_ownership and loan_intent. Using the data in X_cat, we created *X_dummy*, which was the dummy variable encoded version of the X_cat data. We then created the *X_bin* matrix for data that was already binarily encoded, which included loan_status and default_on_file. We then concatenated X_dummy and X_bin to combine all of the now binarily encoded data into the matrix *X_bin_all*. We created our final predictive features dataset by concatenating X_num with X_bin_all, to get *X*. We also performed ordinal encoding on the target data, just to make all of our data in the model numeric, although leaving the target data in loan_grade as letter grades should not have made a difference.

2.3 Train, Test, Split

For the logistic regression and neural network models, the data was split into a training and testing set. We trained on 80% of the original data and performed testing on 20% of the original data. We also shuffled the sets during the split to get a more randomized split of the data. For the SVM model, the loan grade training set was split into training, validation, and test sets. The dataset was split twice to create the training, validation, and testing sets. The first split was between the training/validation set and the test set, and the second split was between the training set and the validation set. For the first split, 7% of the dataset was reserved for testing, while the rest of the data was used for training/validation. For the second split, 93% of the remaining data was set for the training set and 7% was set for the validation set.

2.4 Standardization

The numerical data in the datasets were standardized using z-score normalization. To perform this standardization for the logistic regression and neural network models, we simply subtracted the mean of the X_train dataset from the X_train dataset and divided it by the mean of the X_train dataset. We performed the same z-score normalization on the X_test dataset using the same calculation using the mean and standard deviation of the X_test dataset. For the SVM model, we used SkLearn's StandardScaler() function to perform this standardization on the X_train, X_val, and X_test datasets. In doing so, we made the data in each feature relatively normally distributed. We performed the z-score normalization in hopes of preventing one feature from overly influencing the setting of the training parameters and outcome of the model. In doing so we hoped to reduce bias in the model.

3. Logistic Regression

3.1 One vs Rest Classification with L1 Regularization

In this variation of the logistic regression model, L1 regularization to determine results on the training scores, testing scores, and highest testing accuracy score we could achieve using different c values in "One vs Rest" (OVR) Multi-Class Classification. The OVR algorithm essentially splits the problem into a binary classification problem per class. A binary classifier is trained on each class in the multiclass classification problem and the binary classifier with the most accurate results is chosen to be the representative class in the multiclass classification problem for each example. In our algorithm, we tested multiple *c* values, ranging from 0.0001 to 1000.0, in order to observe the various training and validation scores each penalization factor using L1 regularization resulted in. We kept track of the highest validation score received, and that score's corresponding *c value*. In the figure below we can observe the training and validation scores of each *c* value derived using this model.

C	Training Accuracy	Testing Accuracy
0.0001	0.3254037538192929	0.339909217877095
0.001	0.6864525139664804	0.6864525139664804
0.01	0.7682671322566564	0.7672835195530726
0.1	0.7686163247490179	0.7688547486033519
1.0	0.7689218681798341	0.7683310055865922
10.0	0.7689655172413793	0.7683310055865922
100.0	0.7690091663029245	0.7683310055865922
1000.0	0.7690091663029245	0.7683310055865922

Table 1: Training and Validation Accuracy for OVR with L1 Regularization

We achieved a maximum validation accuracy score of 0.769, at a C of 0.1. However, the maximum testing accuracy score between a C of 0.01 to 1000.0 wasn't that significant, which would suggest that the C for the l1 penalty term did not have a significant effect on the accuracy of the model after a C of 0.01. The training accuracy also remained pretty consistent with the testing accuracy for all C-values suggesting that the OVR model using L1 regularization was a well-fit model and did have a lot of variance. From there we printed out a confusion matrix using the model and the *c* value corresponding to the maximum testing accruacy score. Doing so gave us a visual, comparing the predicted versus actual values of each grade possible in the loan_grade target variable. We also calculated the f1_score to be 0.74, the precision score to be 0.74, and the recall score to be 0.77. From the confusion matrix, we can observe that the model was the most accurate when predicting loans with grades A' and 'B'. For the remainder of the possible loan grades, there were either not many in the dataset itself or our model was incorrect for most of those cases, most likely due to their minimal representation in the dataset. The training accuracy scores in the 0.65-0.78 range would also suggest that there was a fair amount of bias in this model as well.

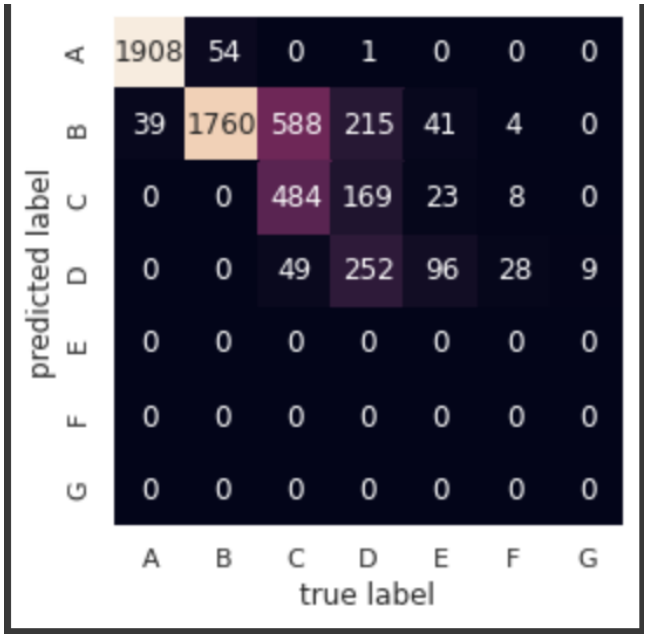

Figure 2: Confusion Matrix for OVR with L1 Regularization

Lastly, when observing our plot of the training and validation curves we can see that the two begin to converge rather quickly for smaller values of *c*, and continue to do so as *c* increases. It is around a *c* value of 10 that we see the two curves are closest to each other, which would suggest that with l1 regularization and a *c* value of 10, we have a relatively well-fitted model. As c increases, the training and validation curves converge rather quickly and remain consistent around a 0.76-0.77 accuracy score, once again suggesting that there was not a lot of variance in the model.

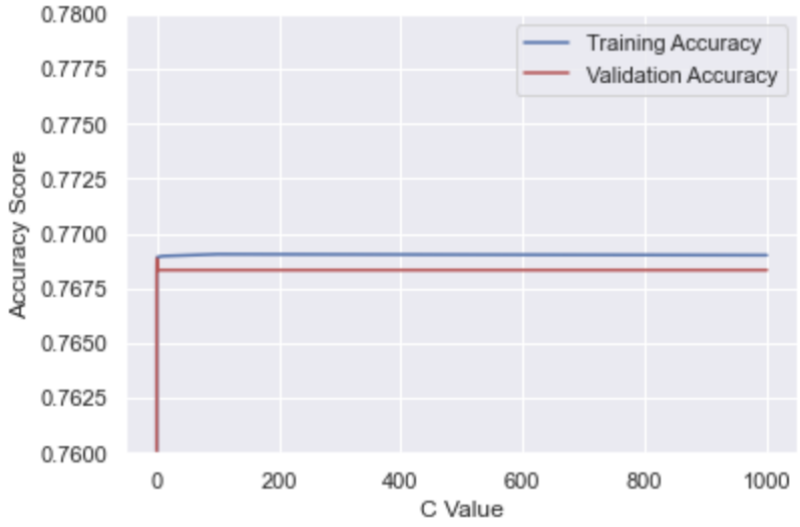

Figure 3: C-Value vs Accuracy Score for OVR with L1 Regularization

3.2 One vs Rest Classification with L2 Regularization

In this variation of the logistic regression model, L2 regularization to determine results on the training scores, testing scores, and highest testing accuracy score we could achieve using different c values in “One vs Rest” Multi-Class Classification. Similarly to our algorithm with L1 regularization, we tested multiple *c* values, ranging from 0.0001 to 1000.0, in order to observe the various training and validation scores each penalization factor using L2 regularization resulted in. We kept track of the highest validation score received, and that score’s corresponding *c value*. In the figure below we can observe the training and testing accruacy scores of each *c* value derived using this model.

C	Training Accuracy	Testing Accuracy
0.0001	0.6414229594063727	0.6393156424581006
0.001	0.7151462243561764	0.7098463687150838
0.01	0.7569183762549105	0.7548882681564246
0.1	0.7675250982103885	0.766236033519553
1.0	0.7685290266259276	0.7683310055865922
10.0	0.7690091663029245	0.7683310055865922
100.0	0.7690091663029245	0.7683310055865922
1000.0	0.7690528153644697	0.7683310055865922

Table 2: Training and Testing Accuracy for OVR with L2 Regularization

We achieved a maximum testing accuracy score of 0.768, at a C of 1.0. In addition, there seemed to be negligible differences in testing accuracy scores for values of C after 0.001 in the model using L2 regularization, with no accuracy score differing by more than roughly 0.02 accuracy points. From there we again printed a confusion matrix, comparing the predicted versus actual values of each grade possible in the loan_grade target and variable. The model was still the most accurate when predicting loans with grades of ‘A’ and ‘B’. This most likely occurred as it did in L1, due to the minimal representation of loan grades past ‘D’ in the dataset. This may have prevented our model from properly training on data classifying loans with grades past ‘D’, thus resulting in less accuracy when classifying them in the testing phase. We also calculated the f1_score to be 0.74, the precision score to be 0.77, and the recall score to be 0.74.

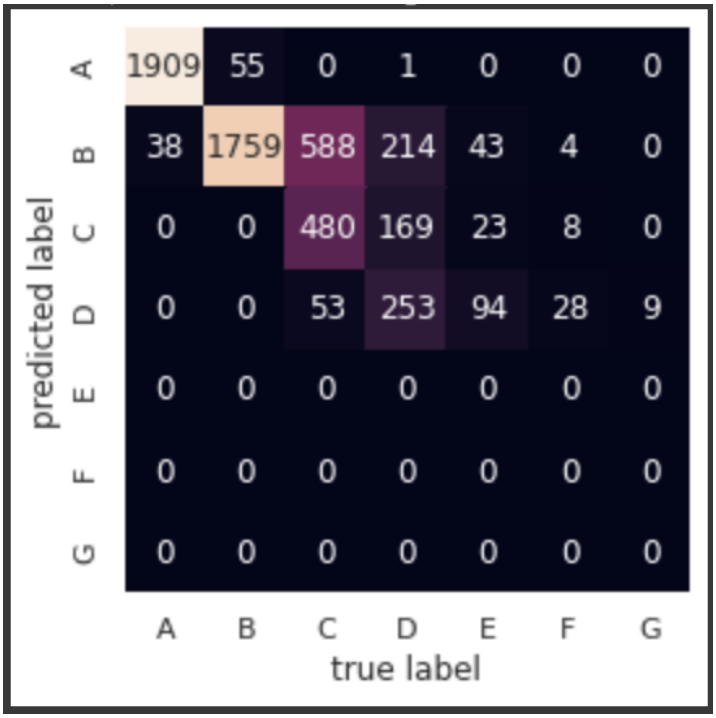

Figure 4: Confusion matrix for OVR with L2 Regularization

When plotting the training and validation curves we can see that the two begin to converge at a similar rate to the model using L1 regularization. In fact, even as c increases and approaches 1000, there still seems to be a consistent difference between the training and validation scores which would suggest low variance in the model. As with using L1 regularization, however, the model using L2 regularization still produces somewhat low accuracy scores in the 0.65-0.78 range, suggesting underlying bias in the model. Thus the behavior of the models between using L1 and L2 regularization were relatively similar with no significant differences.

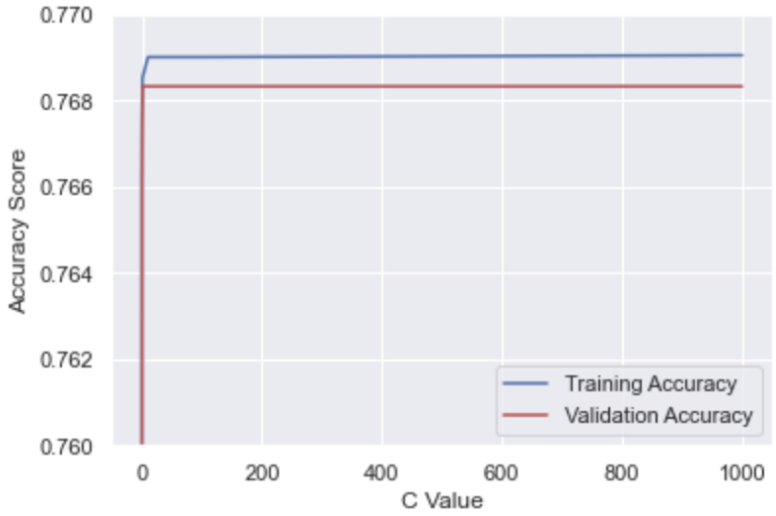

Figure 5: C-Value vs Accuracy Score for OVR with L2 Regularization

3.3 OVR with Polynomial Feature Transformation

After conducting one versus rest classification using both l1 and l2 regularization on the scaled training and test datasets, we decided to see if we could get more accurate or favorable results on the testing data using polynomial feature transformation. To do so, we used sklearn's *PolynomialFeatures* library. This allowed us to initialize a transformation object whose polynomial degree we set to 2. Using this transformation object, we performed a fit and transformation on the X_train and y_train data. We then proceeded to test this transformed data using our one-versus-rest models with L1 and L2 regularizations.

When testing the model trained with the transformed data, we tested using C-values of 0.0001, 0.001, 0.01, and 0.1. We found that with C-values greater than 0.1, the model produced results with accuracy scores that were significantly lower than that of any trials run prior, with training accuracy and testing accuracy scores around 0.3. This would suggest that using C values outside of that range with polynomial feature transformation would lead to underfitting of the data and a significant amount of bias for C-values greater than 0.1. Within this range, however, we saw accuracy score results that the *OVR* models with polynomial feature transformation, perform significantly better than those without.

C	Training Accuracy (L1)	Testing Accuracy (L1)	Training Accuracy (L2)	Testing Accuracy (L2)
0.0001	0.3254037538192929	0.3399092178770095	0.7433871671759057	0.7407472067039106
0.001	0.8270187690964644	0.8308310055865922	0.8027498908773462	0.7973114525139665
0.01	0.8707551287647316	0.8816340782122905	0.859493670886076	0.8606843575418994
0.1	0.8889131383675251	0.8940293296089385	0.8872544740288084	0.8917597765363129

Table 3: Training and Testing Accuracy with Polynomial Feature Transformation (Degree 2)

When using the transformed features, we achieved a maximum testing accuracy of 0.894 at a C of 0.1 with L1 regularization and maximum testing accuracy of 0.892 at a C of 0.1 with L2 regularization. Based on the c-value versus accuracy score graphs below we can see that like without the feature transformation, the training and testing curves converged rather quickly but continued to increase at a faster rate as the c-values increased, compared to the rate at which the accuracy score increased without the feature transformation. Thus based on the maximum and average accuracy scores, we can determine that the model performs better with polynomial feature transformation, with lowered bias as indicated by the higher training and testing accuracy

scores, while maintaining low variance, as suggested bu the relatively consistent values between training and testing accuracy values across all trials.

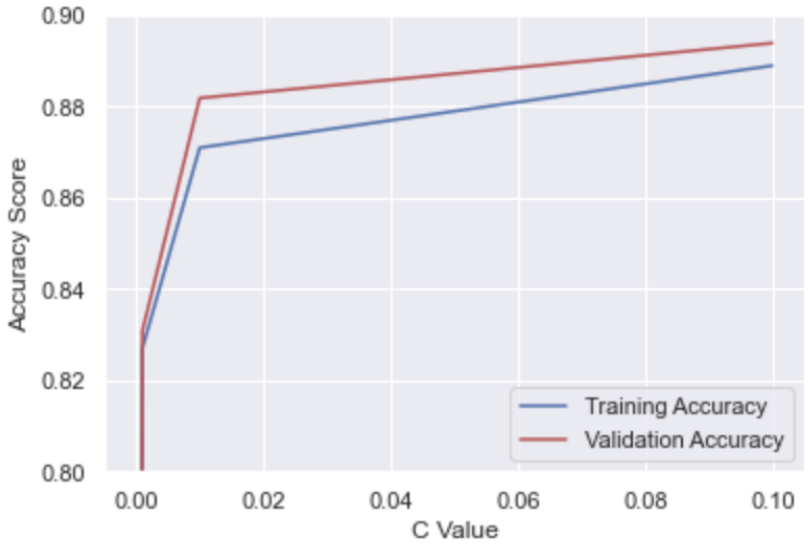

Figure 6: C-Value vs Accuracy Score for Polynomial Feature Transformation on OVR with L1 Regularization

Figure 7: C-Value vs Accuracy Score for Polynomial Feature Transformation on OVR with L2 Regularization

3.4 Logistic Regression/OVR Conclusion

Based on the data and visuals above, we can determine that when creating the multiclass classification logistic regression model, the difference in performance between using L1 regularization and L2 regularization was more or less negligible based on the maximum and average testing accuracy and classification scores. However, we did observe a significant increase in training and testing accuracy when adding polynomial feature transformation to the model training data. This would suggest that the data although the data does perform poorly

without the polynomial feature transformation, it is not necessarily that linearly separable. Thus the polynomial feature transformation of degree helps establish a model that better fits the training and testing data, with lowered bias, while maintaining a low degree of variance which was also present without the polynomial feature transformation.

4.0 Support Vector Machine (SVM)

4.1 Background

The second model that was used on the credit risk dataset to predict loan grades (A-G) was a Support Vector Machine (SVM). This model used the same data preprocessing steps as the logistic regression model. In addition, this model was created using Sklearn's SVM library of functions. The model was tested using SkLearn's model selection library.

After doing research on multiclass classification using SVM, we decided to use the One vs One (OvO) technique over the One vs Rest (OvR) technique, since the models' performances "do not scale in proportion to the size of the training dataset and using subsets of the training data may counter this effect" (Brownlee 2020). This technique was achieved by setting the decision_function_shape parameter of the SVM models to 'ovo'.

Three variations of the SVM model were created, each with a different kernel function: linear, polynomial, and Radial-Basis Function (RBF). These kernel functions were set by changing the kernel parameter to either 'linear', 'poly', or 'rbf'. These three kernel functions allowed us to transform our nonlinear data into a linear space. By using these kernel functions, we avoid using complex computations to create a hyperplane in higher dimensions.

4.2 Baseline SVM Models

	Linear	Polynomial	RBF
Training Accuracy Score	0.889212	0.8506137	0.739947
Testing Accuracy Score	0.901746	0.8488778	0.683292

Figure 8: Accuracy Scores on Baseline Models - prior to Cross-Validation Optimization of Hyperparameters

Before optimizing the model parameters, we set up base models for initial training and testing. The three models were created and trained using the training dataset. The linear model had an initial C value of 0.1. The polynomial model had an initial degree of three and a C value of 0.1 The RBF model had a gamma value of 1 and a C value of 0.1. The training and test accuracy scores were calculated using SkLearn's accuracy_score() function (Figure 8). These scores would be used to compare with the accuracy scores after tuning the C-value, degree, and gamma value parameters using cross-validation.

4.2 Optimization: Testing Model Parameters using Cross-Validation Method

In order to improve the accuracy scores of the three SVM models, the training and validation sets were used to test out different c-values, degrees for the polynomial function, and gamma values for the RBF kernel model.

Originally, in our optimization algorithm, test_param_vals(), the C values tested ranged from 0.0001 to 1000; however, these values were computationally expensive and resulted in low accuracy scores. This resulted in only testing C values in the range 0.001 to 100. This was the same case for the polynomial function degree. The original algorithm tested a polynomial function of degree 10 and 15; however, those computations took too long to run (> 30 minutes). Therefore, the degree values tested were two through five. Lastly, the gamma values tested were 0.1, 1, and 10. These values were chosen based on online resources from SkLearn's documentation.

For each model variation, the C value was tested. As each C-value was tested, a new linear model was created and trained using the training data. The accuracy score for the training set was calculated. Then the model was used to predict the validation set data examples. The accuracy score of the validation set was calculated and recorded. This value would be stored and used to see which C value results in the max validation accuracy score. This C value would be chosen for the optimized version of the model. The results were graphed. This process was replicated for the degree and gamma values.

4.4 Optimization: Linear Kernel Function

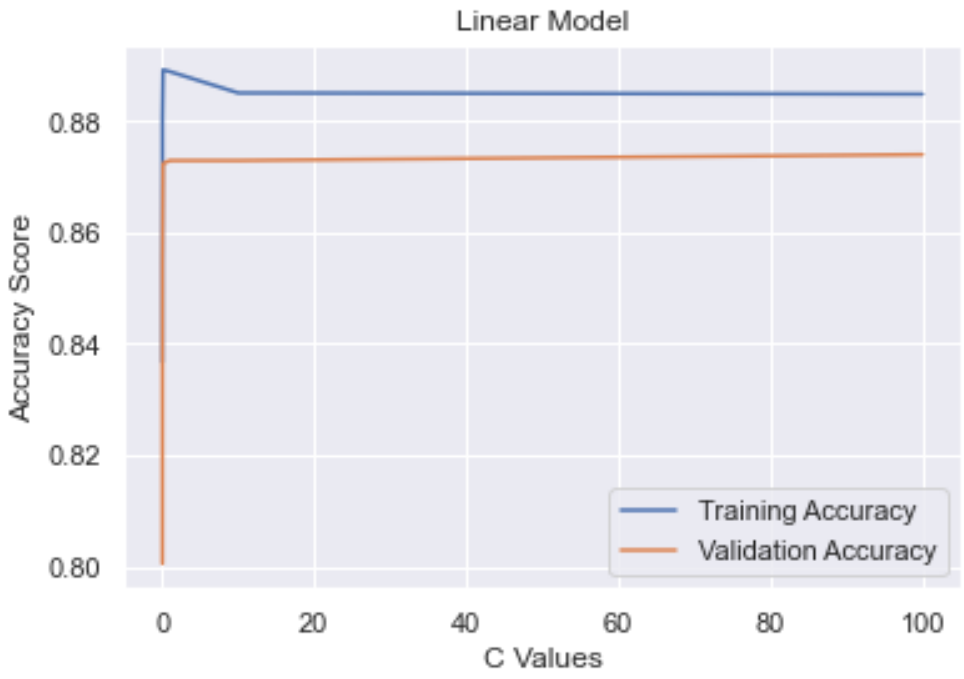

Figure 9: Accuracy Scores of Linear Model Predicting the Training and Validation Sets

The first variation of the SVM model used a linear kernel function. As discussed above, the linear model was tested before and after optimizing the C value hyperparameter. The reason for tuning this regularization parameter is to avoid underfitting/overfitting of the credit risk dataset. During the optimization process, the training and validation scores were calculated and graphed (Figure 9). As expected, the validation accuracy is lower than the training accuracy scores. From the optimization process, the validation scores showed that a linear kernel function model with a C value of 100 produced the highest accuracy score of approximately 0.8740. This value was chosen for the final linear model. In addition, as the C-values increase, the validation accuracy scores show to deviate from the training accuracy scores - leaving a substantial difference between the two towards the end of the C-value range. This could suggest that the linear model overfits the data and cannot generalize well. This furthers the need to investigate nonlinear kernel functions, such as polynomial and RBF functions.

4.5 Optimization: Polynomial Kernel Function

The second variation of the SVM model used a polynomial kernel function. There were two important hyperparameters investigated: degree and C value. The function degree allows us to adjust the complexity of the kernel function, and the C value is used to regularize the function - to avoid underfitting and overfitting.

Training Accuracy	2	3	4	5
0.001	0.351785	0.36313	0.36313	0.360505
0.01	0.736394	0.683826	0.558382	0.549136
0.1	0.85994	0.850614	0.8039	0.74241
1	0.89337	0.896722	0.893007	0.878795
10	0.899629	0.916546	0.930071	0.934876
100	0.902576	0.926801	0.953044	0.967377

Validation Accuracy	2	3	4	5
0.001	0.337802	0.350134	0.331903	0.3437
0.01	0.717426	0.663271	0.5437	0.531367
0.1	0.833244	0.818767	0.768365	0.718499
1	0.862198	0.859517	0.84504	0.812869
10	0.872386	0.860054	0.850402	0.835925
100	0.871314	0.859517	0.835925	0.831635

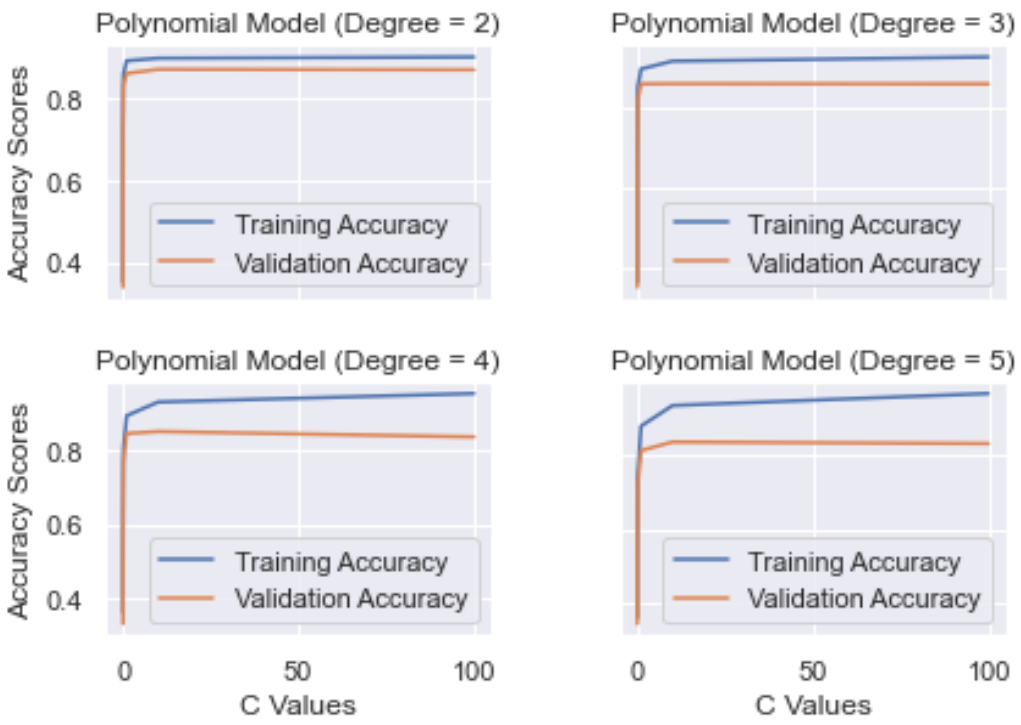

Figure 10: Accuracy Scores of Polynomial Model Predicting the Training and Validation Sets with Varying Degrees

During the optimization process, each degree function was tested among various C values. The training and validation accuracy scores were calculated and graphed. The optimization process illustrated that a polynomial with a degree of 2 fits the data the best with the lowest bias and variance. As the complexity of the polynomial function increased, the gap between the training and validation accuracy scores increased – indicating that the higher degree functions overfit the data and cannot generalize well to new data. The optimization process also showed that with a degree of 2, a C value of 10.0 resulted in the highest accuracy score of 0.8724. These parameters were used for the final polynomial kernel SVM model.

4.6 Optimization: RBF Kernel Function

The third variation used a radial-basis function (RBF) kernel. Similar to the polynomial kernel function, the RBF model transforms nonlinear data into a linear space. However, the RBF model uses another parameter, gamma (γ), along with the regularization parameter, C. The gamma parameter affects how much influence a single training example has, while the C-value often minimizes the effect single training examples have in the prediction decision. These two parameters are traded-off in optimal SVM RBF models. This is why different combinations of C and gamma were tested using the cross-validation method - as discussed above.

Training Accuracy

	0.1	1	10
0.001	0.581516	0.328932	0.328932
0.01	0.746917	0.590197	0.328932
0.1	0.87173	0.739947	0.328932
1	0.900476	0.961765	0.999273
10	0.924459	0.99459	1
100	0.951833	0.999677	1

Validation Accuracy

	0.1	1	10
0.001	0.546381	0.317426	0.317426
0.01	0.771883	0.552279	0.317426
0.1	0.842895	0.685255	0.317426
1	0.869169	0.818767	0.529759
10	0.871314	0.798391	0.543164
100	0.849866	0.794638	0.543164

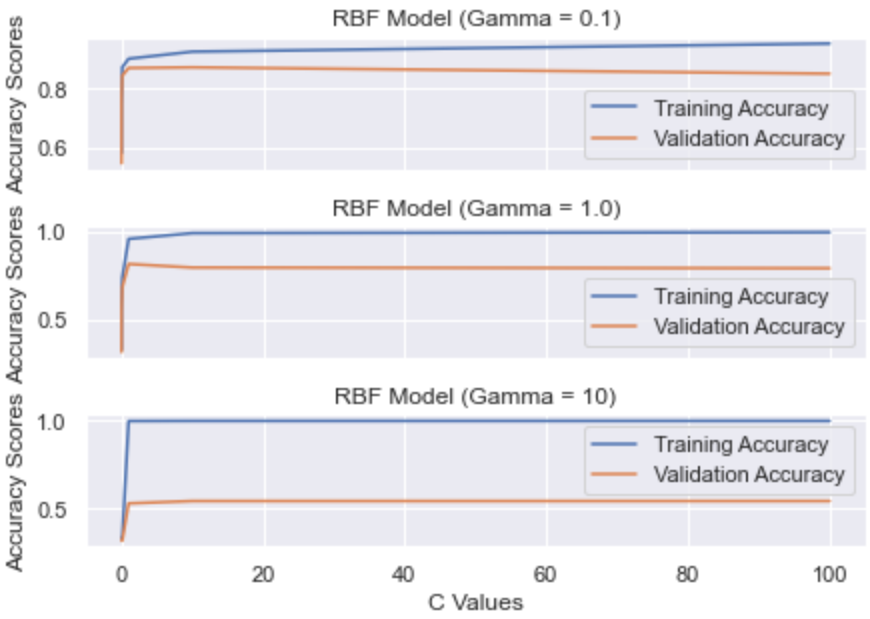

Figure 11: Accuracy Scores of Polynomial Model Predictions with Varying Gamma Values

The results from the RBF optimization process showed that a gamma value of 0.1 was the most optimal as this resulted in greater accuracy scores when predicting the data in the validation set. Furthermore, the optimal C value with a gamma value of 0.1 was 10. This combination of C and gamma resulted in the highest accuracy score of approximately 0.8713 when tested on the validation set. These two values were set in the final RBF model.

4.7 Final Models and Conclusion

	Linear	Polynomial	RBF
Training Accuracy Score	0.884851	0.899628553	0.924459
Testing Accuracy Score	0.897756	0.897755611	0.887282

Figure 12: Accuracy Scores of Final Linear, Polynomial (degree=2), and RBF Models after Optimization

Once the hyperparameter optimization was complete for each SVM variation, the final models were created using the new C, degree, and gamma values. Based on the final results, the optimization technique improved the accuracy scores of the polynomial and RBF model; however, the accuracy score of the linear model decreased by approximately 0.0041. Overall, the polynomial model with a degree of 2 and C value of 10 and the linear model tied in performing the best when introduced to new data - indicating that it generalizes well when compared to the linear and RBF models. However, the polynomial model had a smaller gap between the training

and validation accuracy score, indicating a low variance in the model; therefore, that was chosen to be the optimal SVM model.

5.0 Neural Networks

5.1 Keras Model

In order to create a neural network model to fit our loan data, we used the python *Keras* library. Specifically, we used the Keras *Sequential* model, and to model the hidden layers in the Keras Sequential model, we used the *Dense* model library.

5.2 Base Neural Network Model

In the base version of our neural network model, we established the neural network structure to have one input layer with 20 nodes, one for each feature in the training and test sets, three hidden layers, and an output layer with 7 nodes. The output layer used a 'softmax; activation function to perform the final classification probability calculations. In correlation with the softmax function, the output layer also determined loss using *categorical cross-entropy*. In using categorical cross entropy while training, we penalize the probability of a predicted class based on how far it is from the actual label. The output layer had 7 nodes to model the 7 possible classification outcomes for the loan grade ranging from 'A' to 'G', and to fit the model, these values were dummy variable encoded.

For the hidden layers, we decided to test 3-, 4-, and 5-layer neural network models because based on previous findings, we did not believe our data were linearly separable. For the 3-layer model, we had 2 hidden layers, in which the first stored 24 nodes, and the second stored 16. For the 4-layer model, we had 3 hidden layers, which stored, 24, 16, and 12 nodes in that order respectively. Lastly, for the 5-layer model, we had 4 hidden layers, which stored 24, 20, 16, and 12 nodes in that order specifically. Each hidden layer used the 'Relu' activation function for all models. We then trained each model with the training data and recorded the accuracy and loss scores from the test data.

Number of Layers in Model	Loss Score	Accuracy Score
3	0.280	0.896
4	0.286	0.889
5	0.287	0.889

Table 4: Loss and Accuracy Scores for Base Neural Network Model

5.3 Adding L2 Regularization

For the next step in building upon our neural network model, we added an l2 regularization term to each hidden layer of our neural network, for all models of varying layers.. We did so using the Keras *regularizers* library to import an l2 regularization term for the *kernel_regularizer* hyperparameter in the initialization of our Dense layers. We tested the implementation of l2 regularization using four different regularization terms 0.0001, 0.001, 0.01, and 0.1. Within these trials, the highest accuracy score we achieved was 0.894, using 0.001 as the l2 regularization term, for the 3-layer model. The 4-layer behaved in a relatively similar fashion as its maximum accuracy score also came with a regularization term of 0.001, and the remainder of their scores resembled that of the 3-layer model, although slightly less accurate. The 5-layer model, however, achieved its highest accuracy score with a regularization term of 0.0001 and had a much more drastic drop-off in accuracy as the regularization term increased. These behaviors are easily observable when graphing the 'value' against the accuracy score. We also observed that as the regularization term increases, the accuracy score decreases in a relatively linear fashion. Adding l2 regularization did ultimately help reduce bias and variance on average as the testing accuracy scores increased by a small margin for the most part, while the categorical cross entropy remained around the 0.3-0.35 range.

Number of Layers	Best *l*	Best Loss Score	Best Accuracy Score
3	0.001	.341	.894
4	0.001	.336	.894
5	0.0001	.301	.893

Table 5: Best l, loss score, and accuracy score for Neural Network Model with L2 Regularization

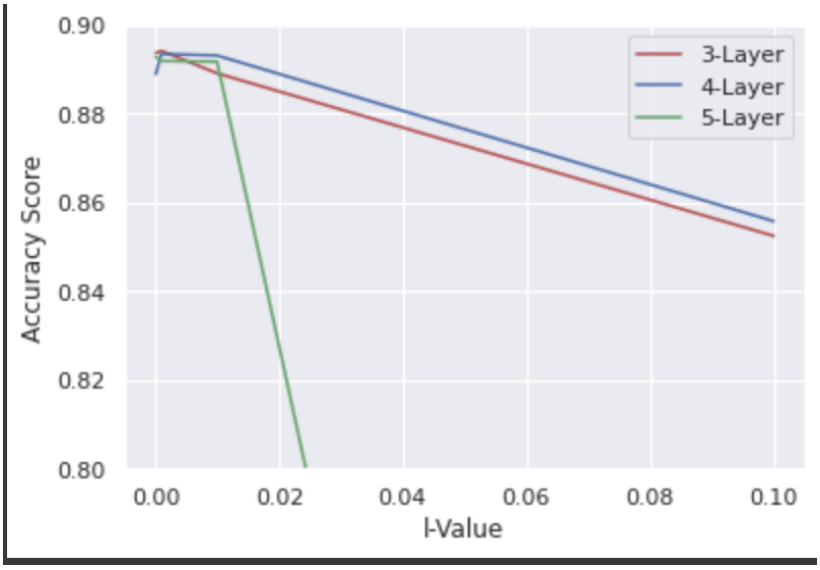

Figure 13: l-value vs Accuracy Score for Neural Network with L2 Regularization

5.4 Implementing the Dropout Regularization

After implementing an l2 regularization term, we also incorporated the dropout behavior for our neural network models. Dropout is another regularization method where we randomly drop out nodes during the training of our model. Having nodes "dropped" during the learning process creates noise, affecting the connectivity and updates of each layer. This forces nodes that have not been dropped to probabilistically taking the responsibility for correcting the mistakes of previous layers. This in theory should help reduce overfitting and generalization error due to the increased robustness and need to learn from randomly dropped nodes storing information. In our models, we incorporate dropouts at varying rates after initializing each hidden layer, apart from the one prior to the output layer. In doing so we trained and tested on incorporated dropout rates ranging from 0.1 to 0.5, in intervals of 0.1, for the 3-, 4-, and 5- layer models.

From our results, we determined that for our model, dropout rates of 0.1 and 0.2 were the most effective in helping reduce generalization and loss error, all the while increasing accuracy. The maximum accuracy we recorded was 0.895, which occurred in the 3-layer and 4-layer models when using a dropout of 0.2. From multiple trials and additional research, we were also able to determine that although both are regularization techniques when used in correlation should generate the least generalization error and produce the best fit model. Based on the results we observed 0.1 to be the best dropout rate across all models, with the maximum accuracy score achieved being 0.897 for the 3-layer, and the best accuracy scores for the 4- and 5-layer models achieved similar results as well. The categorical cross-entropy loss scores remained relatively consistent between the 0.33-0.37 range.

Number of Layers	Best Dropout Rate	Best Loss Score	Best Accuracy Score
3	0.1	0.339	0.897
4	0.1	0.346	0.892
5	0.1	0.357	0.891

Table 6: Best Dropout, loss score, and accuracy score for Neural Network

We can see from the graph below however that the 5-layer model had a much steeper dropoff in its accuracy score as the dropout rate increased past 0.4. These dropoffs were also present in the 3-layer and 4-layer models, but to a much smaller degree, as the accuracy for the 5-layer model dropped to 0.8. Dropout was used in tandem with the l2 regularization in an attempt to minimize the variance and generalization error on the testing data. However, adding dropout did not seem significantly affect the accuracy score relative to how the model performed with only the l2 regularization, albeit with a slight increase in the best accuracy score. Thus adding dropout to this model might help reassure the minimization of the variance/generalization error, even though not significant enough to use it alone or in addition to the l2 regularization.

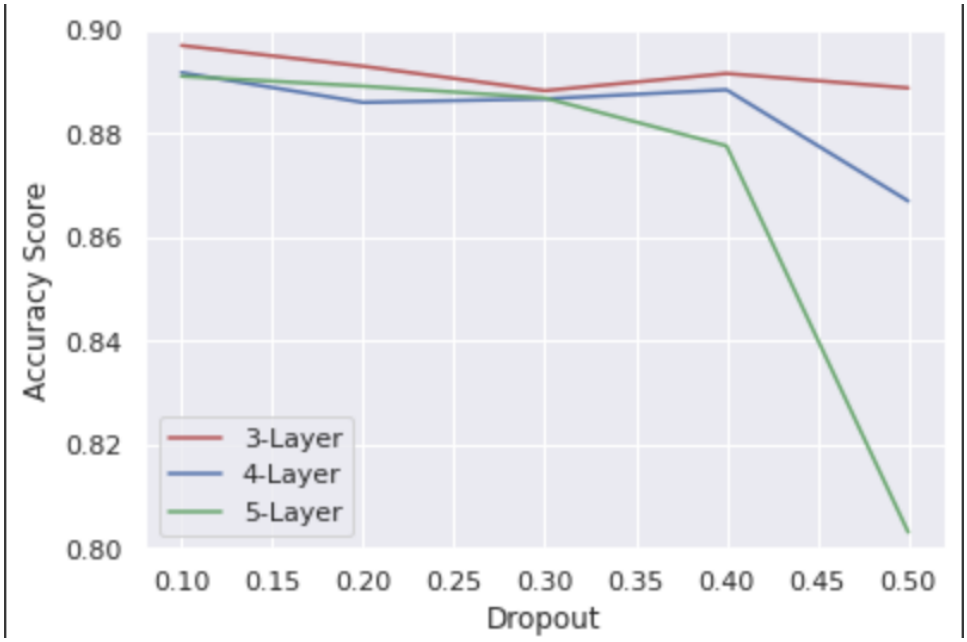

Figure 14: Dropout vs Accuracy Score for Neural Network

5.6 Testing Different Activation Functions

The last major change we attempted to make to our neural network model was changing the activation function. Of the activation functions compatible with our *Dense* hidden layers from the Keras library, we decided to test the use of the Relu, sigmoid, tanh, Selu, and ELU activation functions on all the different layered models. Apart from the sigmoid function, the other activation functions produced relatively similar results in terms of loss and accuracy scores, ranging in the 0.88-0.90 range. Ultimately, we found that Relu, as we had been using, generated the highest accuracy score on average (0.896), although the difference in performance between Relu, tanh, Selu, and ELU was negligible.

Number of Layers	Best Activation Function	Best Loss Score	Best Accuracy Score
3	relu	0.336	0.896
4	elu	0.322	0.896
5	selu	0.339	0.896

Table 7: Best Activation Function, loss score, and accuracy score for Neural Network

5.6 Neural Network Conclusion

In our final neural network model, we accumulated the data and observations we made in our previous altercations of the base neural network model and combined many of these

features/techniques to the model. To start, the use of a 3-layer model produced the most accurate results for the most part and so we decided to use a 3-layer model with hidden layers with 24 and 16 nodes respectively. We incorporated both l2 regularization and dropout in an attempt to minimize generalization errors and prevent overfitting from the complexity of the neural network. We used an l2 term of 0.001 and a dropout rate of 0.2. All in all, we achieved a final accuracy score of 0.894 and a loss of 0.339. However, although there were marginal increases from trying to optimize the model and minimize variance and generalization error using l2 regularization, dropout, and trying different activation functions, the base 3-layer model alone produced accurate results on par with that of the altered models. This would suggest that the 3-layer model alone would be a good fit model for what we have perceived to be a relatively non-linearly separable model.

6.0 Conclusion

	Logistic Regression	SVM	Neural Network
Testing Accuracy	0.8918	0.8978	0.894

Figure 8: Best Testing Accuracy Across all Models

The best model obtained from each model (Logistic Regression, SVM, and Neural Network) were compared. The best Logistic Regression model used a polynomial feature transformation of degree 2, C value of 0.1, and after L2 regularization obtained an accuracy score of 0.892. The best SVM model also used a polynomial kernel function of degree 2 and a C value of 10. This model resulted in a testing accuracy score of 0.8978. Lastly, the optimal neural network model had an architecture of 3 layers - with the first hidden layer having 24 nodes and the second hidden layer having 16 nodes. The three layers (not including the input layer) showed to be the most optimal, while models with 4 and 5 layers showed to overfit the data - with low validation accuracy scores. Even though the difference is marginal, the SVM model with a polynomial (degree=2) kernel function proves to perform the best with a prediction accuracy of 89.78%.

For the SVM model, while multiple degree values were tested, a polynomial function with degree 2 showed to have the best prediction accuracy. This is validated through the results of the logistic regression model, since that model also showed that a polynomial feature transformation of degree 2 led to the best prediction accuracy of loan grades.

Compared to the baseline model, the SVM model increased its prediction accuracy on the test set by 5.33% - as a result of the cross-validation optimization method. This process resulted in the accuracy scores difference between the training and test sets decreasing. The decrease between the accuracy score gap indicates that the variance decreased, as the model is less sensitive to new data. This is ideal for generalization, since the model will not deviate from the high training accuracy scores when introduced to new data. Although not as significant, the

model’s bias decreased slightly, as the difference between the average prediction and correct value decreased. This was validated through a higher training accuracy score - demonstrating that there is less misclassification/error between the average prediction and the true labels.

Bibliography

Brownlee, J. (2019, August 6). *A gentle introduction to dropout for Regularizing Deep Neural Networks*. MachineLearningMastery.com. Retrieved December 7, 2022, from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Brownlee, J. (2021, April 26). *One-vs-rest and one-vs-one for multi-class classification*. MachineLearningMastery.com. Retrieved December 7, 2022, from https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

Brownlee, J. (2022, August 6). *Multi-class classification tutorial with the Keras Deep Learning Library*. MachineLearningMastery.com. Retrieved December 7, 2022, from https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/

Koech, K. E. (2022, July 16). *Cross-entropy loss function*. Medium. Retrieved December 7, 2022, from https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e#:~:text=Categorical%20cross%2Dentropy%20is%20used,%5D%20for%203%2Dclass%20problem.

3.2. Tuning the hyper-parameters of an estimator. (2022). scikit-learn. Retrieved December 7, 2022, from https://scikit-learn.org/stable/modules/grid_search.html#grid-search

RBF SVM parameters. (2022). scikit-learn. Retrieved December 7, 2022, from https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html