Report of Hyperparameter and Pipeline Optimization

Introduction

I wound up combining the two exercises Hyperparameter and Pipeline Optimization. I had messed with hyperparameter optimization a little in the Warmup exercise but removed those to have hyperparameter optimizing be in its own exercise. From the previous exercise the results were not very good, except for Random Forest sometimes reaching ~0.65 (Though not sure if I would really call that good). I wanted to try to bring the other three models up into the 0.60s and if possible, get one of the models to reach 0.7 or greater.

Dataset Description

The dataset used is a red wine dataset, that has eleven distinct features of wine. Those eleven features are: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. There are no missing values in this dataset, and during training, the quality category is removed and used to compare the predicted results to the actual results. There are 1599 different wines in this dataset, and none are missing any values in any of the categories.

Experimental Setup

Starting from where the Alg Selection Exercise left off, I used those Accuracy Estimates as my baseline. I started by researching some basic hyperparameters for each of the models. I defined a dictionary of hyperparameters to test out different combinations.

Baseline Accuracy Estimates:	
Random Forest (RF):	0.6125
SVC:	0.54375
Neural Network:	0.54375
KNN:	0.55625
Test Set Accuracy:	
Random Forest:	0.625

Hyperparameters:

- RF:
  - Number of trees: [50, 100, 200, 500]
  - Depth of the tree: [5, 10, 20, 50]
  - Min # to split a node: [2, 5, 10, 20]
  - Min # to be a leaf: [1, 2, 4, 8]
- SVC:
  - Regularization parameter: [0.01, 0.1, 0.5, 1, 10]
  - Kernel Type: ['linear', 'rbf', 'poly', 'sigmoid']
- NN:
  - # of neurons in a hidden layer:[(10,), (50,), (100,), (200,)]
  - Regularization of L2 Penalty: [0.0001, 0.001, 0.01, 0.1]
- KNN:
  - # of neighbors used: [3, 5, 7, 10]
  - The weight function: ['uniform', 'distance']
  - Power parameter: [1, 2]

Results

The first set of results from running the program with those hyperparameter settings showed a significant increase in the KNN model. The Random Forest stayed the same with both SVC and Neural Network models increasing a few percentages. The next set of changes I made, proved to be a little difficult for my PC to run and I wound up running it on ARCC's HPC to have more processing power. The changes I made were only for NN and KNN. The Neural Network's accuracy estimate increased again, though just a little, while the KNN accuracy estimate dropped a few percentages but still came out on top.

Accuracy Estimates:	
Random Forest:	0.6125
SVC:	0.575
Neural Network:	0.59375
KNN:	0.65625
Test Set Accuracy:	
KNN:	0.66875

Hyperparameters set 2:

- RF:
  - Number of trees: [50, 100, 200, 500]
  - Depth of the tree: [5, 10, 20, 50]
  - Min # to split a node: [2, 5, 10, 20]
  - Min # to be a leaf: [1, 2, 4, 8]
- SVC:
  - Regularization parameter: [0.01, 0.1, 0.5, 1, 10]
  - Kernel Type: ['linear', 'rbf', 'poly', 'sigmoid']
- NN:
  - # of neurons in a hidden layer: [(50,), (100,), (200,), (500)]
  - Solver for weight optimization: ['lbfgs', 'sgd', 'adam']
  - Regularization of L2 Penalty: [0.0001, 0.001, 0.01, 0.1]
  - Learning Rate: ['constant', 'invscaling', 'adaptive']
- KNN:
  - # of neighbors used: [3, 5, 7, 10]
  - The weight function: ['uniform', 'distance']
  - Algorithm: ['ball_tree', 'kd_tree', 'brute']
  - Power parameter: [1, 2]

Accuracy Estimates 2:	
Random Forest:	0.6
SVC:	0.575
Neural Network:	0.6
KNN:	0.625
Test Set Accuracy:	
KNN:	0.6625

Running the training for the Neural Network is what took the longest and the most processing power. My next adjustments was to try to lower the amount of time and processing power needed for the Neural Network, while allowing it still reach 0.6. I got an error several times during the training, about the Neural Network hitting the max iterations, so that was increased from 10,000 to 100,000. I also need to improve SVC, and try to get that model to 0.6.

To run it again on my PC and not on ARCC's HPC I dropped the Neural Network's layer size of 500, and as expected the accuracy estimate also dropped. Random Forest stayed in same range, as did KNN. The unfortunate part is SVC did not change. Meaning that for the next set I will need be trying to adjust several more hyperparameters for SVC to try to improve it to 0.6.

Accuracy Estimates 3:	
Random Forest:	0.61875
SVC:	0.575
Neural Network:	0.56875
KNN:	0.65625
Test Set Accuracy:	
KNN:	0.66875

I once again ran the code on ARCC's HPC to be able to run the Neural Network with up to 500 neurons in a layer. Also, with the addition of several more hyperparameters for tuning in the SVC I knew it would be tough on my PC. Additionally, for the NN it seems that what was affecting it the most was the number of neurons in the hidden layers. The more neurons the higher its accuracy estimate, but also the run time and processing power increased greatly.

Accuracy Estimates 4:
Incomplete, did not finish.

The settings I used for the hyperparameters set 4 were too much and was unable to complete. I ran the program on ARCC's HPC, for five hours over night, and it timed out before completing. Wanting one more run I removed two of SVC's hyperparameters, the degree of the kernel and the gamma setting to try to improve run time.

Accuracy Estimates 5:

Random Forest:	0.61875
SVC:	0.50625
Neural Network:	0.60625
KNN:	0.65625
Test Set Accuracy:	
KNN:	0.66875

Hyperparameters set 3:

- RF:
  - Number of trees: [50, 100, 200, 500]
  - Depth of the tree: [5, 10, 20, 50]
  - Min # to split a node: [2, 5, 10, 20]
  - Min # to be a leaf: [1, 2, 4, 8]
  - Bootstrapping: [True, False]
- SVC:
  - Regularization parameter: [0.01, 0.1, 0.5, 1, 10]
  - Kernel Type: ['linear', 'rbf', 'poly', 'sigmoid']
  - Degree for the kernel: [2, 3, 4, 5]
  - Gamma: ['scale', 'auto']
- NN:
  - # of neurons in a hidden layer: [(50,), (100,), (200,)]
  - Solver for weight optimization: ['lbfgs', 'sgd', 'adam']
  - Regularization of L2 Penalty: [0.0001, 0.001, 0.01, 0.1]
  - Learning Rate: ['constant', 'invscaling', 'adaptive']
- KNN:
  - # of neighbors used: [3, 5, 7, 10]
  - The weight function: ['uniform', 'distance']
  - Algorithm: ['ball_tree', 'kd_tree', 'brute']
  - Power parameter: [1, 2]

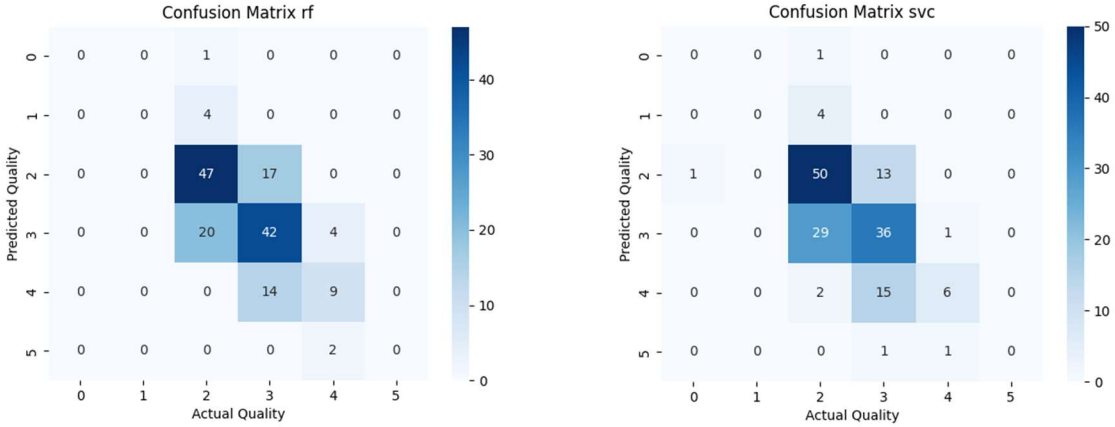
Hyperparameters set 4:

- RF:
  - Number of trees: [50, 100, 200, 500]
  - Depth of the tree: [5, 10, 20, 50]
  - Min # to split a node: [2, 5, 10, 20]
  - Min # to be a leaf: [1, 2, 4, 8]
  - Bootstrapping: [True, False]
- SVC:
  - Regularization parameter: [0.01, 0.1, 0.5, 1, 10]
  - Kernel Type: ['linear', 'rbf', 'poly', 'sigmoid']
  - Degree for the kernel: [2, 3, 4, 5]
  - Gamma: ['scale', 'auto']
  - Independent term: [0.0, 0.1, 0.5, 1.0]
  - Shrinking heuristic: [True, False]
  - Class Weight: [None, 'balanced']
  - Tolerance for stopping: [1e-3, 1e-4, 1e-5]
  - Decision Function Shape: ['ovr', 'ovo']
- NN:
  - # of neurons in a hidden layer: [(50,), (100,), (200,), (500)]
  - Solver for weight optimization: ['lbfgs', 'sgd', 'adam']
  - Regularization of L2 Penalty: [0.0001, 0.001, 0.01, 0.1]
  - Learning Rate: ['constant', 'invscaling', 'adaptive']
- KNN:
  - # of neighbors used: [3, 5, 7, 10]
  - The weight function: ['uniform', 'distance']
  - Algorithm: ['ball_tree', 'kd_tree', 'brute']
  - Power parameter: [1, 2]

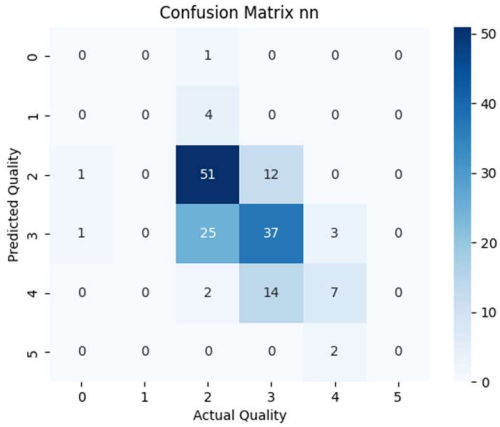
Sadly, the last run of resulted in a worse accuracy estimate for SVC. Random Forest and KNN showed no real change, which makes since due to there not being any change in their hyperparameters. The Neural Network increased again, proving that the larger number of neurons in a layer increases its accuracy estimates for this dataset. I was hoping that I would be able to get SVC up to 0.6, however the settings of the hyperparameters moved its accuracy in the opposite direction. The two hyperparameters I removed were the degree for the kernel, and gamma. The degree is the degree of the polynomial formula and is used to find the optimal hyperplane. I removed this one because using higher degrees increases complexity which can lead to overfitting, and I assumed aided in the immense increase in runtime. The default value for the degree is 3, and looking into this hyperparameter more I have learned that lower degrees can cause a model to underfit, which may have happened here. The second parameter I removed, gamma, may be one of the hyperparameters that may be tuned the most to aid in optimizing performance. I removed it because I was only using 'scale' and 'auto,' and looking more into gamma I learned that 'auto' used to be the default but that was switched to 'scale' in newer versions of the scikit-learn SVC package. Gamma can have non-negative float values manually chosen, though knowing a good float value for the model on a specific dataset is incredibly complicated, and would require many runs to experiment with different tuning settings.

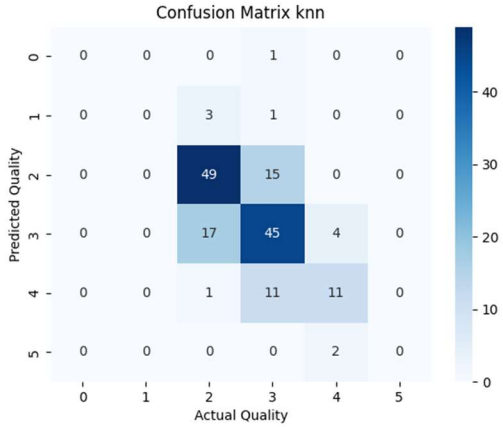
Hyperparameters set 5:

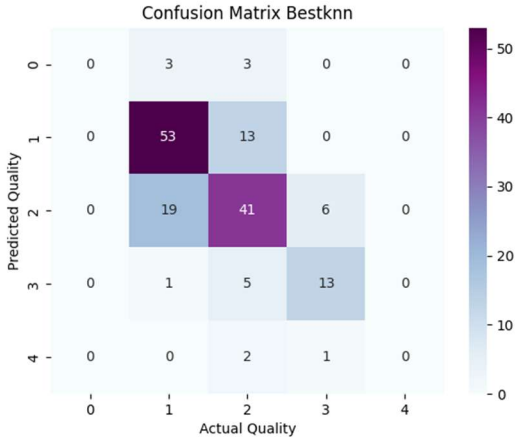
- RF:
  - Number of trees: [50, 100, 200, 500]
  - Depth of the tree: [5, 10, 20, 50]
  - Min # to split a node: [2, 5, 10, 20]
  - Min # to be a leaf: [1, 2, 4, 8]
  - Bootstrapping: [True, False]
- SVC:
  - Regularization parameter: [0.01, 0.1, 0.5, 1, 10]
  - Kernel Type: ['linear', 'rbf', 'poly', 'sigmoid']
  - Independent term: [0.0, 0.1, 0.5, 1.0]
  - Shrinking heuristic: [True, False]
  - Class Weight: [None, 'balanced']
  - Tolerance for stopping: [1e-3, 1e-4, 1e-5]
  - Decision Function Shape: ['ovr', 'ovo']
- NN:
  - # of neurons in a hidden layer: [(50,), (100,), (200,), (500)]
  - Solver for weight optimization: ['lbfgs', 'sgd', 'adam']
  - Regularization of L2 Penalty: [0.0001, 0.001, 0.01, 0.1]
  - Learning Rate: ['constant', 'invscaling', 'adaptive']
- KNN:
  - # of neighbors used: [3, 5, 7, 10]
  - The weight function: ['uniform', 'distance']
  - Algorithm: ['ball_tree', 'kd_tree', 'brute']
  - Power parameter: [1, 2]

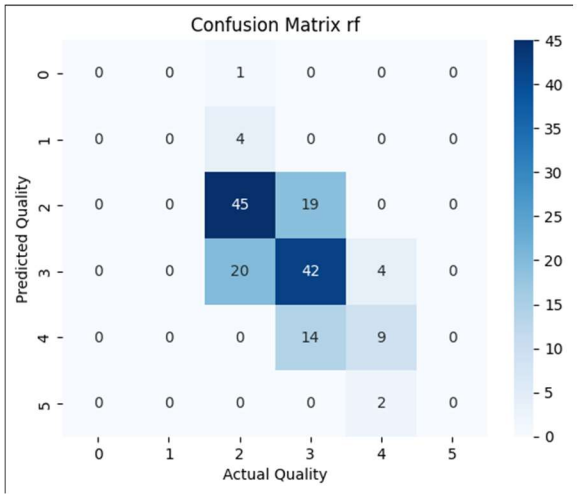

Confusion Matrix rf
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
0 0 47 17 0 0
0 0 20 42 4 0
0 0 0 14 9 0
0 0 0 0 2 0
0 1 2 3 4 5
0 10 20 30 40

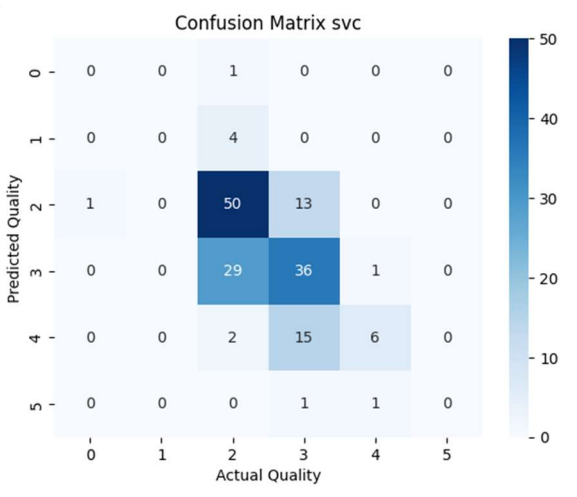
Confusion Matrix svc
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
1 0 50 13 0 0
0 0 29 36 1 0
0 0 2 15 6 0
0 0 0 1 1 0
0 1 2 3 4 5
0 10 20 30 40 50

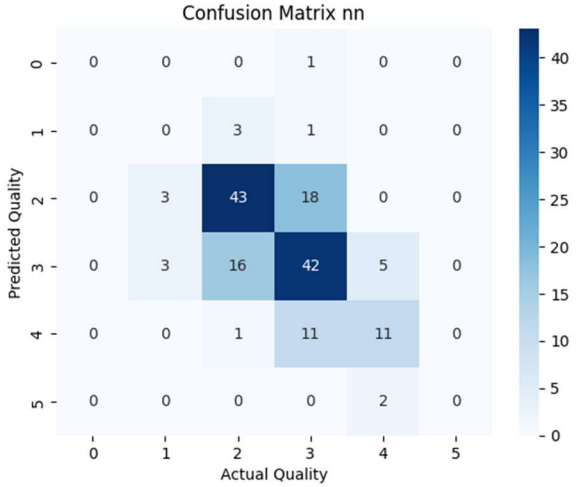

Confusion Matrix nn
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
1 0 51 12 0 0
1 0 25 37 3 0
0 0 2 14 7 0
0 0 0 0 2 0
0 1 2 3 4 5
0 10 20 30 40 50

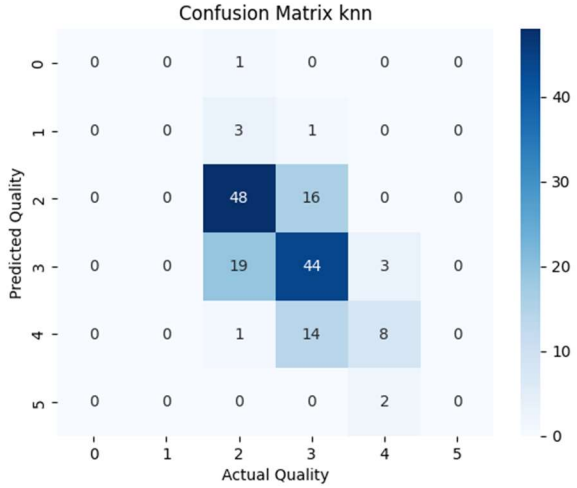

Confusion Matrix knn
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 3 1 0 0
0 0 49 15 0 0
0 0 17 45 4 0
0 0 1 11 11 0
0 0 0 0 2 0
0 1 2 3 4 5
0 10 20 30 40

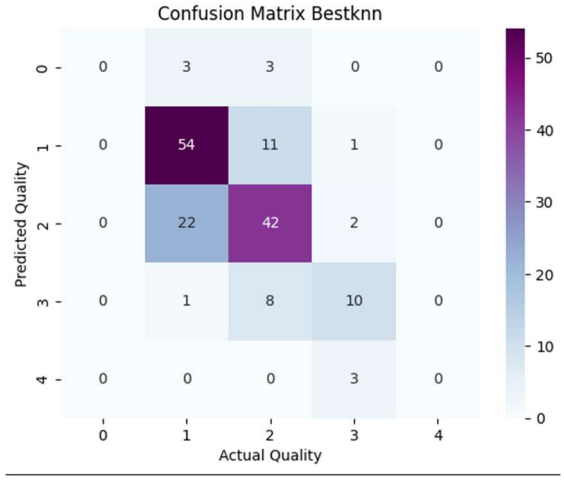

Confusion Matrix Bestknn
Predicted Quality
Actual Quality
0 3 3 0 0
0 53 13 0 0
0 19 41 6 0
0 1 5 13 0
0 0 2 1 0
0 1 2 3 4
0 10 20 30 40 50

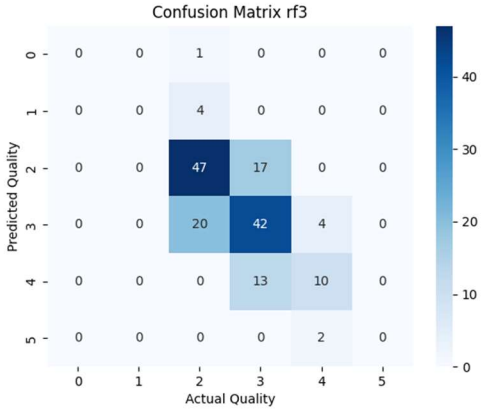

Confusion Matrix rf
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
0 0 45 19 0 0
0 0 20 42 4 0
0 0 0 14 9 0
0 0 0 0 2 0

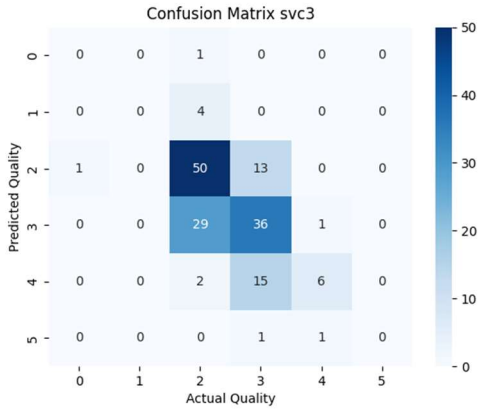

Confusion Matrix svc
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
1 0 50 13 0 0
0 0 29 36 1 0
0 0 2 15 6 0
0 0 0 1 1 0

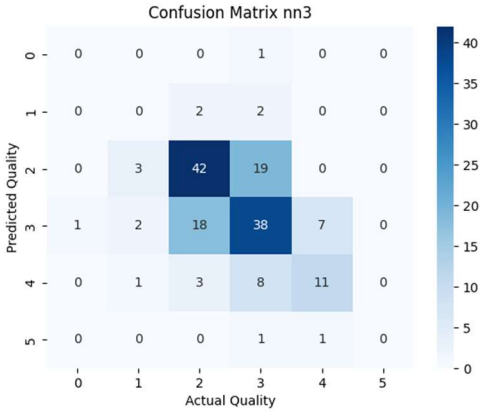

Confusion Matrix nn
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 3 1 0 0
0 3 43 18 0 0
0 3 16 42 5 0
0 0 1 11 11 0
0 0 0 0 2 0

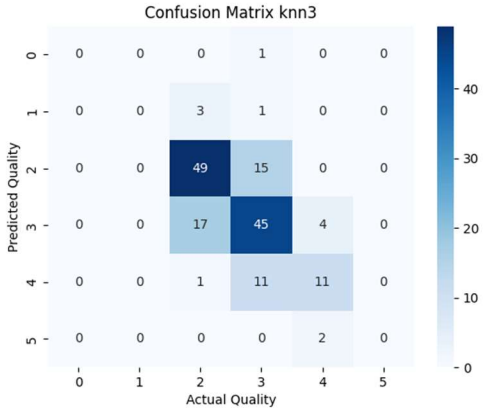

Confusion Matrix knn
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 3 1 0 0
0 0 48 16 0 0
0 0 19 44 3 0
0 0 1 14 8 0
0 0 0 0 2 0

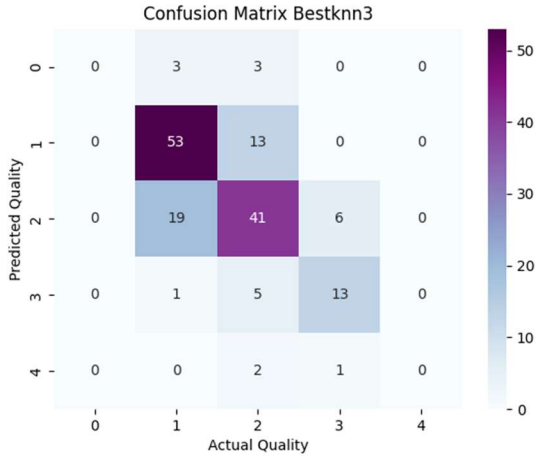

Confusion Matrix Bestknn
Predicted Quality
Actual Quality
0 3 3 0 0
0 54 11 1 0
0 22 42 2 0
0 1 8 10 0
0 0 0 3 0

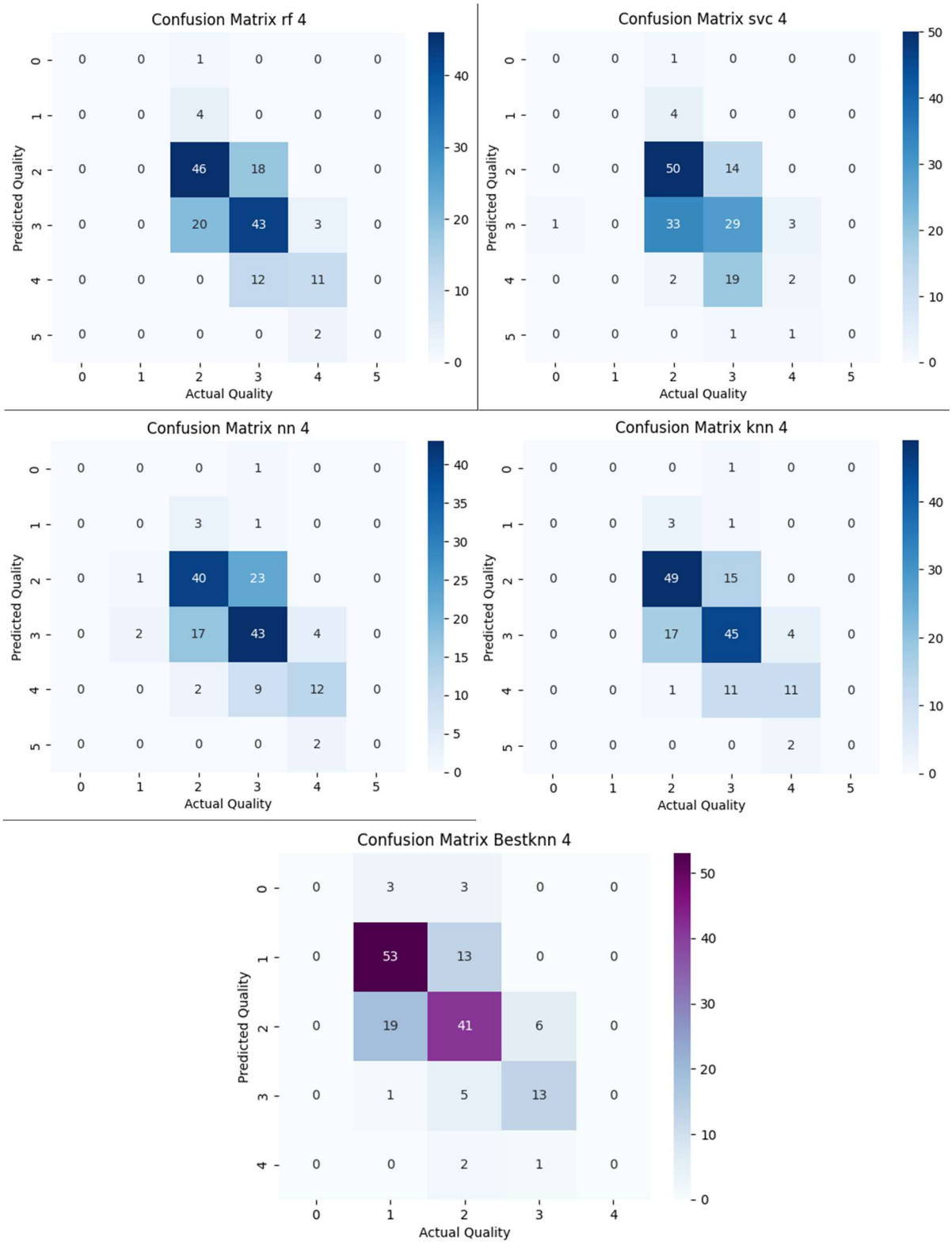

Confusion Matrix rf3
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
0 0 47 17 0 0
0 0 20 42 4 0
0 0 0 13 10 0
0 0 0 0 2 0


Confusion Matrix svc3
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
1 0 50 13 0 0
0 0 29 36 1 0
0 0 2 15 6 0
0 0 0 1 1 0


Confusion Matrix nn3
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 2 2 0 0
0 3 42 19 0 0
1 2 18 38 7 0
0 1 3 8 11 0
0 0 0 1 1 0


Confusion Matrix knn3
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 3 1 0 0
0 0 49 15 0 0
0 0 17 45 4 0
0 0 1 11 11 0
0 0 0 0 2 0


Confusion Matrix Bestknn3
Predicted Quality
Actual Quality
0 3 3 0 0
0 53 13 0 0
0 19 41 6 0
0 1 5 13 0
0 0 2 1 0


Confusion Matrix rf 4
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
0 0 46 18 0 0
0 0 20 43 3 0
0 0 0 12 11 0
0 0 0 0 2 0
Confusion Matrix svc 4
Predicted Quality
Actual Quality
0 0 1 0 0 0
0 0 4 0 0 0
0 0 50 14 0 0
1 0 33 29 3 0
0 0 2 19 2 0
0 0 0 1 1 0
Confusion Matrix nn 4
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 3 1 0 0
0 1 40 23 0 0
0 2 17 43 4 0
0 0 2 9 12 0
0 0 0 0 2 0
Confusion Matrix knn 4
Predicted Quality
Actual Quality
0 0 0 1 0 0
0 0 3 1 0 0
0 0 49 15 0 0
0 0 17 45 4 0
0 0 1 11 11 0
0 0 0 0 2 0
Confusion Matrix Bestknn 4
Predicted Quality
Actual Quality
0 3 3 0 0
0 53 13 0 0
0 19 41 6 0
0 1 5 13 0
0 0 2 1 0