Pipelines and Hyperparameter Optimization

Introduction

My first major attempt at this exercise I did not really do much for the pipelines. I used scikit-learn's StandardScalar to standardize the features in my dataset to make it so the features were easily able to be compared to each other. I needed to go back and add more for the pipelines to do to be of use. The dataset being used is the red wine dataset, and I included hyperparameter optimization (HPO) in this exercise trying to combine pipeline optimization and HPO to more quickly find a way to get higher accuracy.

Dataset Description

The dataset used is a red wine dataset, that has twelve distinct features of wine. Those twelve 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

Picking a few different models to test on, I went for Random Forest (RF) and K-Nearest Neighbors (KNN), due to being able to get high accuracies with these two previously. I chose a Neural Network (NN), MLPClassifier, because from previous experiments I learned you can get good accuracy from a NN, though they typically require a high-performance cost. Lastly, I chose Support Vector Classification (SVC). SVC, I have struggled to get good accuracies out of and wanted to see if I could change that through optimization of pipeline and hyperparameters. The dataset was split into an 80% training set, 10% validation set, and the model that scored the highest accuracy, was ran on the last 10% which was the final test set.

The previous feedback I received stated that I did not have multiple steps in my pipeline, so my first goal was to add more steps. In the previous experiment, I only used one step in my pipeline and that was scikit-learn's StandardScalar to preprocess the dataset features to be standardized. Looking at some of the other student's pipeline optimization code, I saw many were additionally using SimpleImputer. Following their lead, I created a new pipeline that used SimpleImputer and StandardScalar, and then called that pipeline from my pipelines for each model. However, after adding it I saw no change and investigated what it does on scikit-learn's website and learned that it replaces any missing values in the dataset with whatever it is setup as, in this case the median value. For this red wine dataset there are no missing values, which I confirmed by running the library pandas isnull() command to create a printout of the sum of all missing values in each feature, and all totaled to zero missing values. I decided to leave this in still, thinking it was good practice for datasets that are missing values, which can be common. Those two preprocessing steps are for numerical data, and I noticed several students had preprocessing steps for categorical data. I did not add any categorical preprocessing due to the wine quality datasets only using numerical data.

During one of the Graduate Student's presentations, he showed graphs of running the models using different numbers of the features. I thought this was interesting, especially since I had previously used a correlation graph to determine the top five most important features and only used those five. To try to incorporate that student's idea I used another scikit-learn library RFECV (Recursive Feature Elimination with Cross-Validation) to start at using all 11 features (total 12 features minus quality feature) and eliminate one feature at a time that is determined to be the least important. I did this in only the RF and SVC pipelines, since when I tried to for the NN and KNN I got errors caused from those models not being able to use feature selection like the Random Forest and SVC to determine and eliminate a feature.

Each of the four different models I am using additionally each have their own pipeline. I set it up this way so each model can use the best settings for that model. All four models go through and run with the dataset after being standardized. The pipeline for RF and for SVC contains the RFECV to eliminate a feature each iteration to find the most optimal number of features for this dataset.

For hyperparameter optimization I created a dictionary of parameters for each model and used Grid Search to try different combinations of hyperparameter settings. First the model starts in the pipelines to have the data preprocessed, RF and SVC also go through the feature selection that eliminates one feature per iteration. Then that model goes into the grid of hyperparameters for that specific model to try out different hyperparameter settings. I determined which settings to use from scikit-learn's website for that specific model and searched websites and asked AI like ChatGPT questions to learn what each one was in more detail than scikit-learn's website explained. Through a combination of searching online, and experimentation I settled on a few due to computational power requirements primarily, with performance being secondary.

Preprocessing and Hyperparameters:

Random Forest:
imputer__strategy: ['mean', 'median', 'most_frequent']
scaler: [StandardScaler(), MinMaxScaler()]
n_estimators: [50, 100]
max_depth: [5, 10, 20]
min_samples_split: [2, 5, 10]
min_samples_leaf: [1, 2, 4, 8]

Support Vector Classifier:
imputer__strategy: ['mean', 'median', 'most_frequent']
scaler: [StandardScaler(), MinMaxScaler()]
C: [0.001, 0.01, 0.1, 1, 10]
kernel: ['linear', 'rbf', 'poly', 'sigmoid']
gamma: ['scala', 'auto']

Neural Network:
imputer__strategy: ['mean', 'median', 'most_frequent']
scaler: [StandardScaler(), MinMaxScaler()]
hidden_layer_size: [(50,), (100,)]
solver: ['lbfgs']

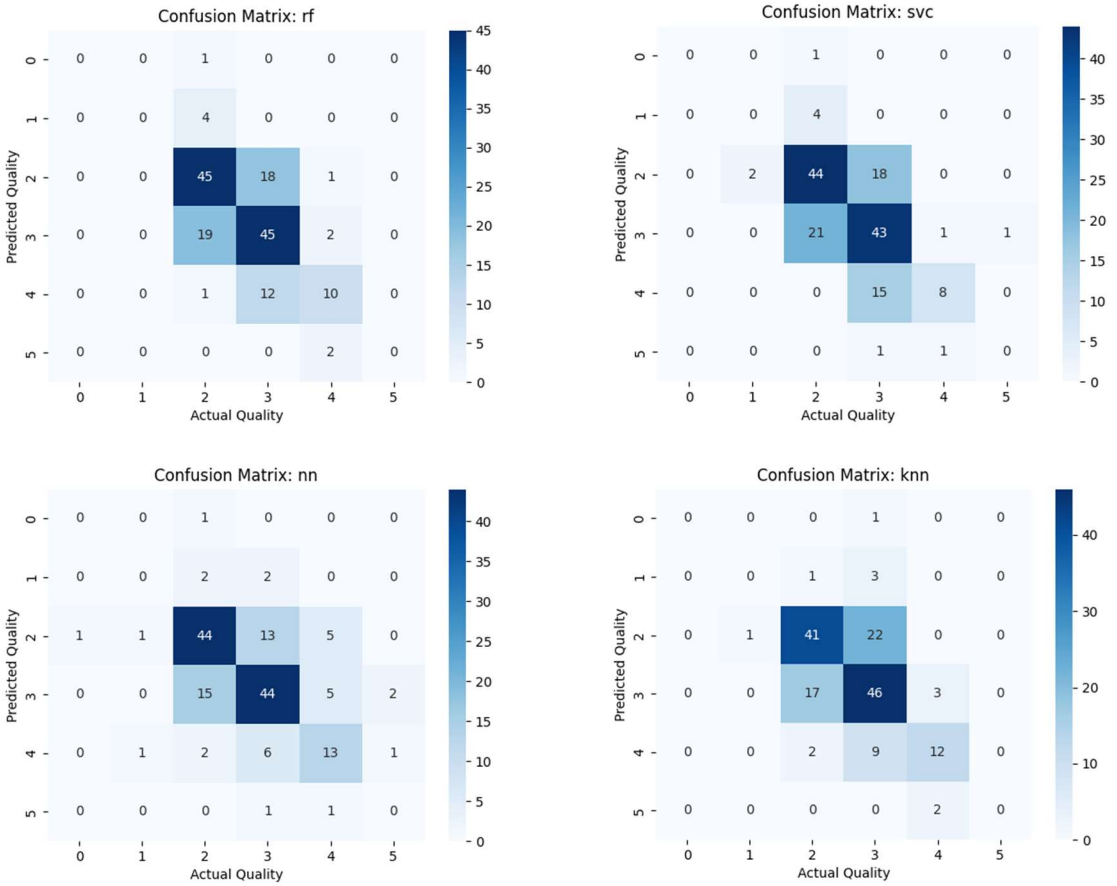
K-Nearest Neighbors:
imputer__strategy: ['mean', 'median', 'most_frequent']
scaler: [StandardScaler(), MinMaxScaler()]
n_neighbors: [3, 5]
weights: ['uniform', 'distance']
algorithm: ['brute']
p: [1, 2]

Results

Throughout setting up the pipeline and adjusting hyperparameter settings I got many different results. Once getting the rough hyperparameters chosen the accuracies stayed roughly around the same, typically only varying a few percent. However, SVC was difficult to get up near 0.6, most of the time it stayed around 0.525. Using the pipeline and the final HPO settings above, these are final results I got, with the highest accuracy being ran an additional time on the test dataset.

	Accuracy
Random Forest	0.6375
Support Vector Classifier	0.59375
Neural Network	0.625
K-Nearest Neighbors	0.625
Best (Random Forest) on TEST set	0.7

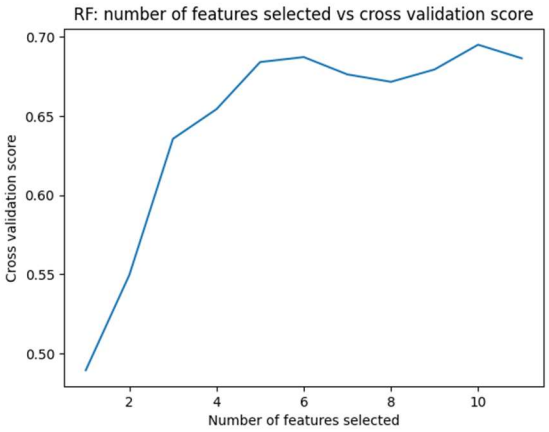
Confusion Matrices of the models:

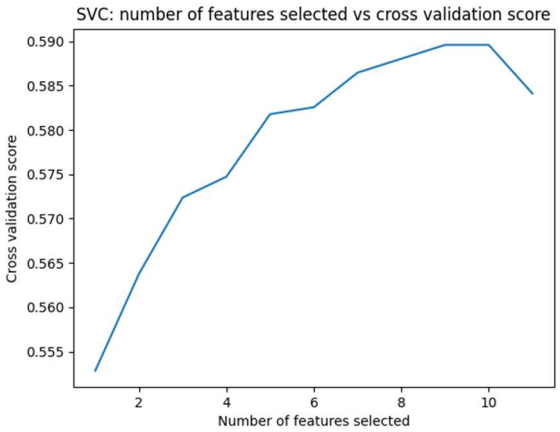


*The 4 at the end of the title is left over from testing, and just forgot to change the Titles when plotting final graphs.

* The Best graph only has quality 0-4, due to the test set just happening to not have 6 categories for quality, and the matrices were made to auto size based of the number of qualities.

Graphs showing the comparison between the cross-validation score and number of features for the Random Forest and SVC models.





These final graphs so the mean scores with different hyperparameter settings. The numbers for the Hyperparameter settings are for the iteration of each different combination of hyperparameters.