Introduction:

For this exercise I experimented with different preprocessing methods and their effect on various ML models as well as with and without hyperparameter optimization. I went with numerical preprocessors because every categorical preprocessor I tried had extremely long runtimes that I would just cancel after 15+ minutes of running.

Dataset Description:

I used the wine quality dataset[1] for this and picked the white wine dataset because it has more entries than the red dataset. This data includes 11 features and a target value called “quality” that is between 0 and 10.

[1]: https://archive-beta.ics.uci.edu/dataset/186/wine+quality

Experimental Setup:

For these experiments I picked a few models to test on, Random Forest because it has been the most accurate in every test on the wine dataset, K Nearest Neighbors because I wanted to how much preprocessing would improve it, and Support Vector Machine because I wanted a third model, and it was used in all the examples on scikit-learn’s documentation.

Then I picked a couple preprocessing methods, I decided to use two different scaler methods because I thought those would have the most impact and I used the basic scaler and a minmax scaler.

Then I ran some tests on all the models with default parameters and optimized parameters with and without preprocessing and compared the results.

The hyperparameter optimizations will both use cross validation with 5 folds.

The hyperparameters optimized are:

Random Forest:

max_depth:

The default value for this is None, and I tested this with values 25, 50 and None, this parameter sets the max depth of the tree.

n_estimators:

The default value for this is 100, and I tested this with values 100, 125 and 150, this parameter sets how many random trees the algorithm will generate.

min_samples_split:

The default value for this is 2, and I tested this with values 2, 4, 6, this parameter determines the minimum amount of samples before a node can split.

min_samples_leaf:

The default value for this is 1, and I tested this with 1, 3, 5, this parameter determines the minimum number of samples in a leaf.

K-Nearest-Neighbors:

n_neighbors:

The default value for this is 5, and I tested this with values 3, 4, 5, 6, 10, this parameter sets the number of neighbors.

leaf_size:

The default value for this is 30, and I tested this with values 10, 30, 50, 100, 200, this parameter sets the leaf size passed into the balltree and kdtree algorithms.

Algorithm:

The default value for this is ‘auto’ and I tested this with 'ball_tree', 'kd_tree', 'brute', 'auto', this parameter changes which algorithm is being used.

P:

The default value for this is 2, and I tested this with both 1 and 2, it changes which distance formula is used.

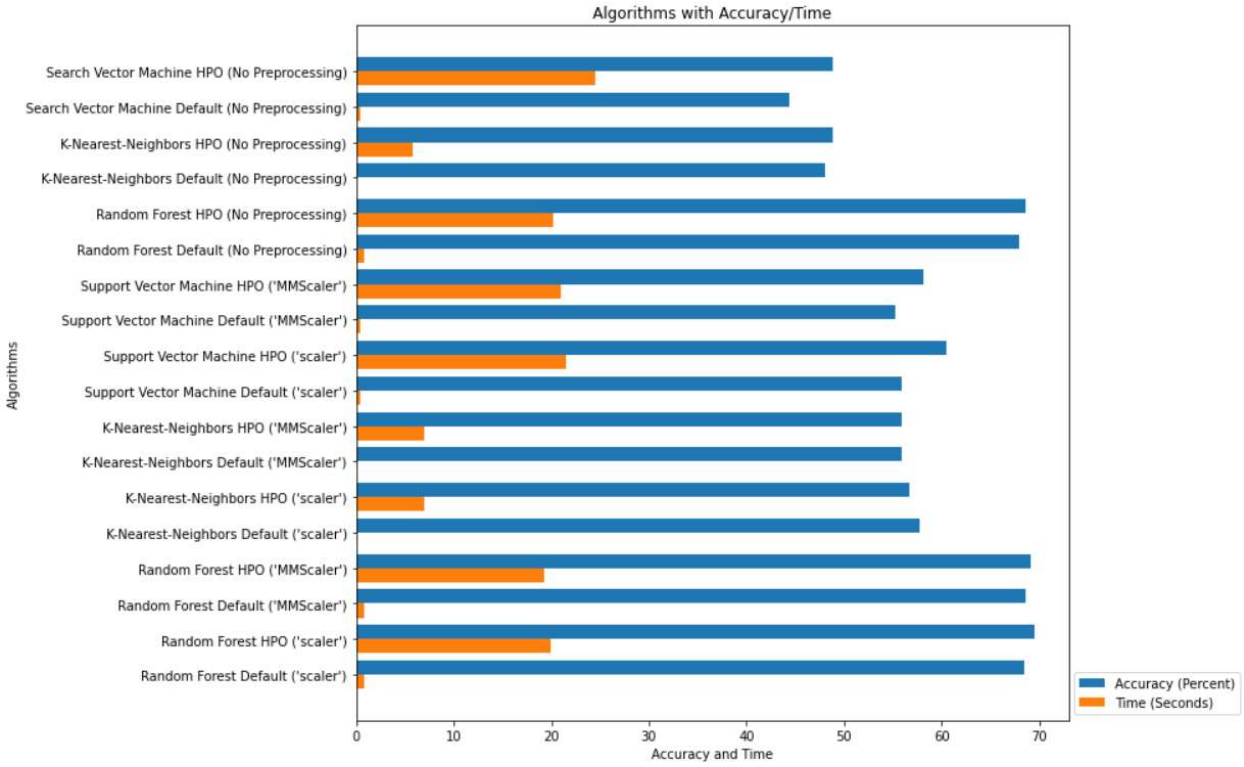
Support Vector Machine:

C:

The default value for this is 1, I tested this with values 1, 10, 100, this parameter changes the regularization parameter.

I then output the results sorting by accuracy first and time second then labeling the best one.

Results:



Best Algorithm by accuracy and time is: Random Forest HPO ('scaler')

List of algorithms sorted best to worst:
Random Forest HPO ('scaler'): 69.48980% accuracy, 19.97628 seconds
Random Forest HPO ('MMScaler'): 69.08163% accuracy, 19.27015 seconds
Random Forest Default ('MMScaler'): 68.57143% accuracy, 0.86925 seconds
Random Forest HPO (No Preprocessing): 68.57143% accuracy, 20.18962 seconds
Random Forest Default ('scaler'): 68.46939% accuracy, 0.89427 seconds
Random Forest Default (No Preprocessing): 67.95918% accuracy, 0.87175 seconds
Support Vector Machine HPO ('scaler'): 60.51020% accuracy, 21.45210 seconds
Support Vector Machine HPO ('MMScaler'): 58.06122% accuracy, 21.00678 seconds
K-Nearest-Neighbors Default ('scaler'): 57.75510% accuracy, 0.01051 seconds
K-Nearest-Neighbors HPO ('scaler'): 56.73469% accuracy, 6.92203 seconds
K-Nearest-Neighbors Default ('MMScaler'): 55.91837% accuracy, 0.00951 seconds
Support Vector Machine Default ('scaler'): 55.91837% accuracy, 0.39934 seconds
K-Nearest-Neighbors HPO ('MMScaler'): 55.91837% accuracy, 7.04793 seconds
Support Vector Machine Default ('MMScaler'): 55.20408% accuracy, 0.39884 seconds
K-Nearest-Neighbors HPO (No Preprocessing): 48.87755% accuracy, 5.86948 seconds
Search Vector Machine HPO (No Preprocessing): 48.77551% accuracy, 24.44761 seconds
K-Nearest-Neighbors Default (No Preprocessing): 48.06122% accuracy, 0.00801 seconds
Search Vector Machine Default (No Preprocessing): 44.38776% accuracy, 0.41085 seconds

List of best hyperparameters per algorithm:
Best Paremeters found for Random Forest ('scaler'): OrderedDict([('classifier__max_depth', 50), ('classifier__min_samples_leaf', 1), ('classifier__min_samples_split', 2), ('classifier__n_estimators', 150)])

Best Paremeters found for Random Forest ('MMScaler'): OrderedDict([('classifier__max_depth', 25), ('classifier__min_samples_leaf', 1), ('classifier__min_samples_split', 2), ('classifier__n_estimators', 150)])

Best Paremeters found for K-Nearest-Neighbors ('scaler'): OrderedDict([('classifier__algorithm', 'kd_tree'), ('classifier__leaf_size', 30), ('classifier__n_neighbors', 3), ('classifier__p', 2)])

Best Paremeters found for K-Nearest-Neighbors ('MMScaler'): OrderedDict([('classifier__algorithm', 'ball_tree'), ('classifier__leaf_size', 30), ('classifier__n_neighbors', 10), ('classifier__p', 1)])

Best Paremeters found for Support Vector Machine ('scaler'): OrderedDict([('classifier__C', 100)])

Best Paremeters found for Support Vector Machine ('MMScaler'): OrderedDict([('classifier__C', 100)])

Best Paremeters found for Random Forest (No Preprocessing): OrderedDict([('classifier__max_depth', 25), ('classifier__min_samples_leaf', 1), ('classifier__min_samples_split', 2), ('classifier__n_estimators', 150)])

Best Paremeters found for K-Nearest-Neighbors (No Preprocessing): OrderedDict([('classifier__algorithm', 'auto'), ('classifier__leaf_size', 10), ('classifier__n_neighbors', 5), ('classifier__p', 1)])

Best Paremeters found for Search Vector Machine (No Preprocessing): OrderedDict([('classifier__C', 100)])

The results show some improvements for random forest with preprocessed data and huge improvements for the support vector and k nearest neighbors algorithms when the data is preprocessed. Also that performance is improved across the board when the data is preprocessed for this dataset.

Code:

Main.py